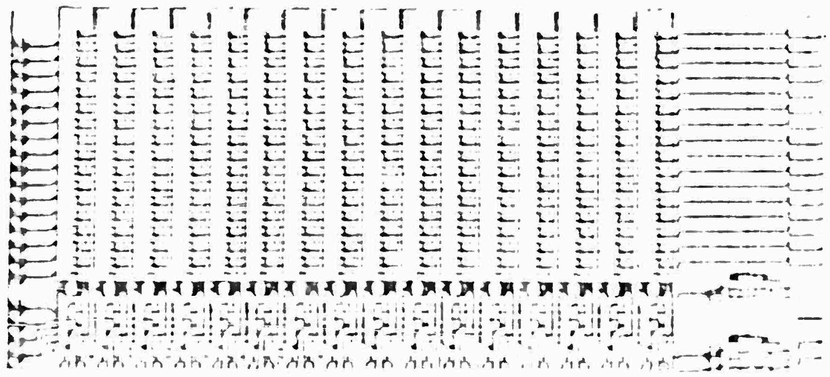
PHOTOGRAPH

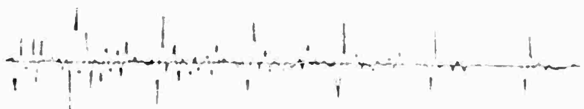
Proceedings Of WORKSHOP ON VLSI & MODERN SIGNAL PROCESSING

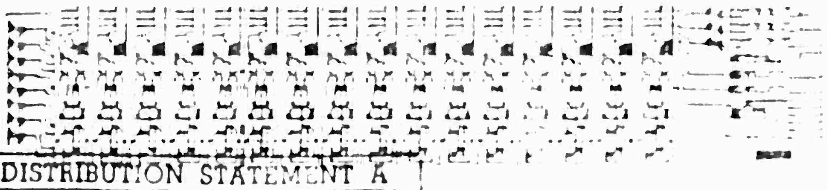
USC

SUN-YUAN KUNG, editor

November 1-3, 1982







COMPONENT PART NOTICE

THIS PAPER IS A COMPONENT PART OF THE FOLLOWING COMPILATION REPORT:

(TITLE): Proceedings of USC (University of Southern California) Workshop on VLSI (Very Large Scale Integration) & Modern Signal Processing, Held at Los Angeles, California on 1-3 November 1982.

(SOURCE): University of Southern California, Los Angeles.

TO ORDER THE COMPLETE COMPILATION REPORT USE AD-A136 855.

THE COMPONENT PART IS PROVIDED HERE TO ALLOW USERS ACCESS TO INDIVIDUALLY AUTHORED SECTIONS OF PROCEEDINGS, ANNALS, SYMPOSIA, ETC. HOWEVER, THE COMPONENT SHOULD BE CONSIDERED WITHIN THE CONTEXT OF THE OVERALL COMPILATION REPORT AND NOT AS A STAND-ALONE TECHNICAL REPORT.

THE FOLLOWING COMPONENT PART NUMBERS COMPRISE THE COMPILATION REPORT:

AD#:	TITLE:
P002 601	Signal Processing Applications of Systolic Array Technology.
P002 602	A Thousand-Word Speech-Recognition System Using Special-Purpose MOS-LSI.
P002 603	A Parallel VLSI (Very Large Scale Integration) Architecture for a Digital Filter Using a Number Theorectic Transform.
P002 604	Frequency Domain Digital Filtering with VLSI (Very Large Scale Integration).
P002 605	Concurrent Algorithms as Space-Time Recursion Equations.
P002 606	A Formal Derivation of Array Implementations of FFT (Fast Fourier Transform) Algorithms.
P002 607	Problem-Oriented Specification of Concurrent Algorithms.
P002 608	An Overview of Signal Representations in Signal Processing Languages.
P002 609	Multi-Computer Parallel Arrays, Pipeline Arrays, and Pyramids.
P002 610	Parallel Implementation of Likelihood-Based Estimation and Detection.
P002 611	Digital Signal Processing Structures for VLSI (Very Large Scale Integration).
P002 612	Linear or Square Array for Eigenvalue and Singular Value Decompositions?
P002 613	Signal Processing in High Data Rate Environments--Design Tradeoffs in the Exploitation of Parallel Architectures and Fast System Clock Rates: An Overview.
P002 614	New Principle of Array Processing in Underwater Passive Listening.
P002 615	High Resolution Spectrum Analysis by Dominant Mode Enhancement.
P002 616	State of the Art in Eigenvalue Computations.
P002 617	Impact of VLSI (Very Large Scale Integration) on Modern Signal Processing.

COMPONENT PART NOTICE (CON'T)

AD#:	TITLE:
P002 618	A Systolic Signal Processor for Recursive Filtering.
P002 619	An Approach to Reliable Parallel Data Processing.
P002 620	Yield Enhancement by Fault Tolerant Systolic Arrays.
P002 621	The Impact of VLSI (Very Large Scale Integration) on Signal Processing.
P002 622	VLSI (Very Large Scale Integration) Architectures for Recognition of Context-Free Languages.
P002 623	Parallel Algorithms for Image Analysis.
P002 624	Concurrent VLSI (Very Large Scale Integration) Architectures for Two Dimensional Signal Processing Systems.
P002 625	VLSI (Very Large Scale Integration) Architectures for Pattern Analysis and Image Database Management.
P002 626	Some Critical Issues in VLSI (Very Large Scale Integration) Signal Processor Implementation.
P002 627	A Comparison of Arithmetic Units in VLSI (Very Large Scale Integration) Signal Processing Systems.
P002 628	The Microelectronics Center: A New Force in Cooperative VLSI (Very Large Scale Integration) Signal Processing Research.
P002 629	Computing Theory and Practice.
P002 630	VLSI (Very Large Scale Integration) - Basis for Cooperation.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE

EE5101	AD-A136 855
(Modern Signal Processing [illegible] and Parallel Computation) Workshop Title: "VLSI and Modern Signal Processing"	FINAL REPORT 011082 [illegible]
Various Sun-Yuan Kung, editor	N00014-83 G-003[illegible]
University of Southern California University Park Los Angeles, California 9000[illegible]	NR049-[illegible]
Office of Naval Research 1030 East Green Street Pasadena, California 91106	15 November 1983
	21[illegible]
[illegible] LIS Associate Director for Eng'g. Sciences Office of Naval Research [illegible] North Quincy Street Arlington, VA 22217	UNCLASSIFIED

Approved for release: distribution unlimited

II. Parallel processors: algorithms, languages, and architectures:

This category is subdivided into (1) systolic processors, (2) Fault tolerant computing structures, (3) reconfiguration and partitioning (4) numerical algorithms and arithmetics and (5) language and formal descriptions.

III. VLSI and Signal processing:

As the central theme of the workshop, this issue has been, at least partially, addressed by all the papers. However, there are several of them more directly confronting the issue.

USC WORKSHOP ON
VLSI AND MODERN SIGNAL PROCESSING

November 1 -3, 1982

Davidson Conference Center

University of Southern California

Los Angeles, California 90089

<u>Co-Chairmen:</u>

S.Y. Kung, University of Southern California,

H.J. Whitehouse, Naval Ocean Systems Center,

T. Kailath, Stanford University,

<u>Organizing Committee:</u>

K. Bromley, Naval Ocean Systems Center

R. Chellappa, University of Southern California,

T. Kailath, Stanford University,

H.T. Kung, Carnegie-Mellon University,

S.Y. Kung, University of Southern California,

A. Laub, University of Southern California,

H.J. Whitehouse, Naval Ocean Systems Center,

J. Speiser, Naval Ocean Systems Center,

TABLE OF CONTENTS

FOREWORD

VLSI microelectronics technology and the emerging computer-aided design methodologies are spurring a new revolution in signal processing. The area of VLSI signal processing is bound to become a major focus of attention in governmental, industrial, as well as university research activity. To provide a bridge from signal processing theory and algorithms to VLSI processor architectures and implementation, it is critical to have a fundamental understanding of the basic computational requirements of modern signal processing and of the technology constraints of VLSI. This will require a cross-fertilization of the fields of computer software/hardware and signal processing engineering. A workshop on this cross-disciplinary field will provide an opportunity for leading research scientists from government, industry, and university to review the general areas of VLSI signal processing, to further enhance the unification of such research activities and to define the future directions for VLSI applications to signal processing.

MODERN SIGNAL PROCESSING AND VLSI

Does modern signal processing need VLSI? The answer is a firm yes. The ever-increasing demands for performance, sophistication and real-time signal processing strongly indicate the need for tremendous computation capability, in terms of both volume and speed. The availability of low cost, high density, fast VLSI devices promises the practicality of cost-effective, high speed, parallel processing of large volumes of data. This makes feasible ultra high throughput-rates and presages major technological breakthroughs in real-time signal processing applications.

Does VLSI need signal processing? At the first glance, the need will appear rather uncertain. On the other hand, it is quite obvious that the full potential of VLSI can be realized only when its application domains are discriminatingly identified. For this purpose, it may be noted that traditional computer architecture designs are no longer suitable for the design of highly concurrent VLSI computing processors. As a major technological constraint, communication in VLSI systems has to be highly restricted, as communication is expensive in terms of area, power and time consumption. Fortunately, most signal processing algorithms have attractive properties of regularity, recursiveness and locality in data-dependence, and thus are very amenable to VLSI implementation. Therefore, the potential of today's fast growing VLSI technology will probably be best exploited by its extensive application to modern signal processing; that is to say, signal processing will be very important for VLSI systems research.

In the past, major work has been done on mapping various kinds of signal processing applications into specific LSI or VLSI architectures. The insight and experience gained from this have already greatly enhanced the understanding of VLSI's impact on signal processing.

Unfortunately, such contributions have been very scattered owing to the lack of interaction between different disciplines. To meet the challenge of matching VLSI and modern signal processing, it is critical to have a fundamental and cross-disciplinary understanding of the basic computational requirements and the technology constraints. Therefore the need for effective information exchange activities such as joint research projects or advanced workshops has become imminent.

In summary, modern technological innovations in VLSI research and increasingly demanding needs in modern signal processing have provided a basis for a very constructive research cooperation between the VLSI and signal processing communities. We note that there have recently been many encouraging developments and successes in the joint VLSI and signal processing research activities. Therefore, we are convinced that a new field, that may be aptly called VLSI signal processing, is emerging. With fairly balanced representation from university, industry, and government, this workshop should offer an opportunity for prompt technology transfer. This should, in the short run, stimulate new or joint research activities; and, in the long run, set up important new research directions.

PROGRAM OVERVIEW

The broad range of the presentations in the workshop can roughly be divided into three main categories:

I. Signal Processing Methods and Arcnitectures

This category is subdivided into: (1) Modern signal processing, including those by Kailath, Levy, Bienvenu and Mermoz, and Owsley; (2) Signal processing architectures, by Troung, et al., Roberts, Tufts, and the speech processing architectures by Buric, Murveit and Broderson; and (3) Image processing, recognition and analysis, by Uhr, Fu et al., Hwang, and Rosenfeld; and (4) Implementation of signal processors, by Swartzlander, Gilbert, Woods et al., Nash and Nudd and Powell .

II. Parallel Processors: Algorithms, Languages and Archictectures

This category is subdivided into (1) Systolic Processors, including those by H.T. Kung, Travassos, and Schreiber and Kuekes; (2) Fault Tolerant Computing Structures, by Kuhn, and Meyer and Weinert; (3) Reconfiguration and Partitioning, by Snyder, and Heller; (4) Numerical Algorithms and Arithmetics, by Ahmed, Lau, Parlett, and S.Y. Kung and Gal-Ezer; and (5) Language and Formal Descriptions, by Chen and Mead, Johnsson and Cohen, Cremers and Hibbard, and Kopec.

III. VLSI and Signal Processing

As the central theme of the workshop, this issue has been, at least partially, addressed by all the papers. However, there are several of them more directly confronting the issue, including those by Mead, Whitehouse et al., Fair, Sumney, and S.Y. Kung.

In the meeting, especially in the panel and informal discussion sessions, we expect all the participants to add much broader insight as well as controversy to this topic.

ACKNOWLEDGEMENTS

From the very beginning, this workshop has been fortunate to have the encouragement and support of the academic, industrial and government communities. The general concept of the workshop was developed in a series of meetings and debates among the members of the organizing committee. Special credits go to Professor H.T. Kung of CMU and Professor R. Chellappa of USC for their contributions to the fine structure of the program. We have benefited greatly from Professor Carver Mead, especially by his cheerful optimism toward the prospect of the future VLSI Signal Processing research. He has also been a valuable source of many fine suggestions for our preparation work. The burden of handling the enormous amount of paperwork and correspondence fell mostly on the shoulders of Linda Varilla of USC. Without her total dedication, the smooth operation we have so far enjoyed would not have been possible. On behalf of the workshop co-chairmen, Dr. H. J. Whitehouse, Prof. T. Kailath and myself, I wish to express our thanks to them and to all the authors, panelists, and session chairmen. Finally, the support of the Office of Naval Research is gratefully acknowledged.

Sun-Yuan Kung

Co-Chairman

USC Workshop on VLSI and
Modern Signnal Processing

MODERN SIGNAL PROCESSING

Thomas Kailath
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

ABSTRACT

In modern signal processing, unlike what is called "digital signal processing", the operations we perform are dictated by the application of an optimization criterion, deterministic or statistical. Such approaches often suggest appropriate 'macro' building blocks for implementing the optimal solutions, rather than merely adapting classical analog filters to digital operation, or rather than always performing a Fast Fourier transformation.

There are two major aspects of modern signal processing:

- Determining optimal algorithms
- Implementing the optimal algorithms

and a further goal is to have a proper interaction between these two aspects: implementation considerations being able to influence the form of algorithms, and the nature of the algorithms being able to suggest the form of implementation.

Moreover, for real-time and adaptive operation, we need to be able to do both of these quickly (with "fast" algorithms), recursively (to easily incorporate new data), and cheaply (with special architectures and with special technologies). We hope to demonstrate that new ways of thinking about signal processing algorithms can allow direct translation into hardware, reducing the need for extensive software development.

Among the results discussed will be new constant-gain lattice filters for nonstationary processes and high speed, pipelined *or* parallel, cascade digital structures for pole-zero transfer functions.

SIGNAL PROCESSING APPLICATIONS OF SYSTOLIC ARRAY TECHNOLOGY

(extended summary)

H.J. Whitehouse, J.M. Speiser, and K. Bromley

Naval Ocean Systems Center
San Diego, California 92152

ABSTRACT

Architectures and algorithms are examined for exploiting the large number of degrees of freedom available in current VLSI and projected VHSIC integrated circuit technologies to provide real-time implementations for sonar signal processing tasks.

Such real-time signal processing tasks present a heavy computational load in the form of linear and multilinear transforms including beamforming and crossambiguity calculation; matrix multiplication for covariance matrix estimation, least squares solutions for adaptive processing, and eigensystem solution for high resolution direction finding.

The large number of degrees of freedom will frequently require not only parallelism in the computation, but also that special attention to be paid to the organization of the data flow, memory architecture, and propagation of required control information.

This paper examines the matrix computation requirements for signal processing tasks including adaptive filtering, ambiguity function computation, recent methods of spectrum analysis and direction finding. Parallel processing architectures for implementating these computations are compared, with emphasis on recent developments in systolic architectures providing modular parallelism, local interconnects, and regular data flow for the major signal processing computations.

INTRODUCTION

It has previously been shown that the major computational requirements for many important real-time signal processing tasks can be reduced to a common set of basic matrix operations[1]. These include matrix-vector multiplication, matrix-matrix multiplication and addition, matrix inversion, solution of linear systems, least squares approximate solution of linear systems, eigensystem solution, generalized eigensystems solution, and singular value decomposition of matrices. For implementation on a single processor, the results of extensive research in numerical linear algebra is a set of numerically stable, well documented routines for linear systems solution and least-squares problems - LINPACK [2] and the [illegible] eigensystem problems - EISPACK [3]. Parallel processor designs can utilize the available studies of the numerical stability of algorithms. Parallel processing architectures have been surveyed[1] and it was concluded that the [illegible] architectures [4] provide the most promising [illegible] [illegible]

Previously reported systolic architectures [4] include linear array configurations for matrix-vector multiplication and solution of linear equations with triangular coefficient matrix, and hexagonal configurations for matrix multiplication-addition and L-U decomposition of matrices. These architectures are particularly attractive when the matrices have only a few bands occupied about the main diagonal.

This paper will discuss new architectures for the multiplication of dense matrices, configurations for partitioned matrix operations, and applications to crossambiguity function calculation. Companion papers will discuss a systolic testbed using current microprocessor components [5] and progress in systolic architectures for the least squares and eigensystem problems [6].

LEAST SQUARES PROBLEMS

Three types of least squares problems will be considered: the deterministic problem, the stochastic problem with known second moments, and the stochastic problem with estimated second moments. It will be shown that the stochastic problem with estimated second moments is computationally identical to a deterministic least squares problem, and thus a baseline method for real-time solution in many signal processing applications may be implemented by forming and solving the normal equations via systolic matrix multiplication, matrix-vector multiplication, and matrix inversion. Alternative methods for the least squares problem are known [11] and a systolic implementation is discussed in [65].

The deterministic least-squares problem is the selection of a vector x to minimize ϵ_1^2 in eqn. (1), where A is a known matrix and y is a known vector.

$$\epsilon_1^2 = \|Ax-y\|^2 = x^TA^TAx - 2y^TAx + \|y\|^2 \qquad (1)$$

The superscript T denotes the transpose of a matrix or vector.

The stochastic least squares problem is the selection of a deterministic weight vector w to minimize the expected squared error ϵ_2^2 in eqn. (2) where z is a random vector and d is a random variable. The scalar random variable w^Tz is a linear combination of the elements of the random vector z, and is used as a linear estimator of the random variable d.

$$\epsilon_2^2 = E(w^Tz-d)^2 = w^TR_{zz}w - 2R_{dz}^Tw + [illegible] \qquad (2)$$

Statistical expectation is denoted by the operator E, and R_{zz} [illegible] the second moment matrix of the random vector z: R_{zz} = [illegible]. Similarly, the vector R_{dz} is defined as R_{dz} = $E(dz)$.

[illegible]

[illegible] it will be [illegible] the random variable [illegible] $i = 1, \ldots$.

The standard unbiased estimators of the second moments are shown in eqns. (3 - 5). The corresponding error term for the estimation of d by w·z is shown in eqn. (6).

$$(\hat{R}_z)_{s_1,s_2} = (1/N) \sum_{n=1}^{N} z(n,s_1)\, z(n,s_2) \quad , \tag{3}$$

$$(\hat{R}_{dz})_{s_1} = (1/N) \sum_{n=1}^{N} d(n)\, z(n,s_1)$$

$$= (1/N) \sum_{n=1}^{N} z^T(s_1,n)\, d(n) \tag{4}$$

$$(\widehat{Ed^2}) = (1/N) \sum_{n=1}^{N} d^2(n) \tag{5}$$

$$\epsilon_3^2 = w^T \hat{R}_z w - 2\hat{R}_{dz}^T w + (\widehat{Ed^2}) \tag{6}$$

The correspondence shown in table 1 may then be used to show the computational equivalence of the stochastic least squares problem with estimated second moments and the corresponding deterministic least squares problem.

Deterministic	Stochastic with estimated second moments
$(A)_{ns}$	$N^{-.5}\, z(n,s)$
$y(n)$	$N^{-.5}\, d(n)$
$x(s)$	$w(s)$
$A^T A$	$\hat{R}_z$
$y^T A$	$\hat{R}_{dz}^T$
$\lVert y \rVert^2$	$E(\widehat{d^2})$

Table 1. Desired identifications for computational equivalence of least squares problems.

It is therefore only necessary to consider computational methods for the deterministic least squares problem.

[illegible]

[illegible]

[illegible]

element after propagation through the ship's hull.

The interference cancellation problem is similar to the noise cancellation one, except that the inputs to the adaptive combiner are the outputs of preformed conventional beams. To form one beam with interference cancellation, one conventional beam is steered in the desired-look direction, and S "null beams" are generated. The null beams have zero sensitivity in the desired-look directions, and are used to provide interference references. In general, as many linearly independent null beams will be required as interfering sources to be cancelled. If the beam outputs are combined with fixed weight for the beam steered in the desired-look direction, minimizing the total power output minimizes the interference contribution to the output. If the interference is uncorrelated with the desired signal, the signal contribution to the output power remains constant. Under certain conditions, however, this assumption can be violated: under conditions of multipath propagation, the signal may be received on a null beam as well as on the beam in the desired-look direction. In this case, adjusting the weights for minimum total power output can reduce the signal contribution as well as the noise.

Noise and interference cancellation have been typically implemented via gradient descent adaptive transversal filters [7] - the Widrows LMS algorithm. Such implementations are designed to provide adaptations with a reduced multiplication rate and hence, relatively simple hardware. Unfortunately, the adaptation rate is reduced if there is a large spread in the eigenvalue distribution of the data covariance matrix [9]. Unfortunately, a strong interfering source will result in large eigenvalue spread and may result in a convergence time which is greater than the stationarity time of the problem. In such a context, faster convergence can be obtained via direct inversion of the sample covariance matrix to solve the normal equations [9], making more statistically efficient use of the available data at any give time.

Recent method of spectrum analysis provide improved resolution by utilizing a parametric model of the signal. In the maximum entropy method [10], the signal is modeled at the output of an all-pole filter driven by white noise. The inverse of such a filter is a transversal filter which converts the signal to white noise and whose tap weights are calculated by by solving the linear one-step prediction problem for the signal. It is well known that for a stationary process, the prediction error of the least-squares linear predictor is a white noise process. Then the estimated spectral density function is proportional to the reciprocal of the magnitude squared of the transfer function of the prediction error filter. This spectrum estimation technique, which provides improved resolution when the model is applicable and the signal-to-noise ratio is sufficiently high, has also been applied to beamforming [11] .

The least squares problems corresponding to the noise cancellation, interference cancellation, and maximum entropy spectrum analysis are summarized in Table 1.

Problem	z	d
noise cancellation	$z_1,\ldots, z_S$ (noise references)	signal + noise
interference cancellation via adaptive combiner with fixed beams	$b_1,\ldots\ldots b_S$ (outputs of preformed null beams)	b_0 (output of preformed beam steered in desired look direction)
maximum entropy spectrum analysis	$x_1,\ldots\ldots x_S$ (time samples of signal)	x_{S+1}

Table 2. Correspondences for some applications of least squares solutions.

It will be shown in the full paper that the new systolic architecture called the "engagement processor" provides efficient multiplication of dense matrices. Two types of enhancements to engagement processors were described: a memory with three-dimensional organization for storing matrix intermediate computation terms, and a "bus expander" for loading a row or column of the engagement processor in parallel. Enhanced engagement processors were shown to efficiently perform partitioned matrix multiplication and inversion, one- and two-dimensional discrete Fourier transforms, and crossambiguity function calculation.

REFERENCES

[1A] Speiser, J.M. and H.J. Whitehouse, "Architectures for Real-Time Matrix Operations", Proceedings of the 1980 Government Microcircuits Applications Conference held at Houston, Texas, 19 - 21, Nov. 1980

[1B] Speiser, J.M., H.J. Whitehouse, and K. Bromley, "Signal Processing Applications for Systolic Arrays", Record of 14th Asilomar Conference on Circuits, Systems and Computers held at Pacific Grove, California, 17 - 19, Nov. 1980, IEEE Catalog No. 80CH1625-3, pp. 100 - 104.

[2] Dongarra, J.J., et al, LINPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1979.

[3] Garbow, B.S., et al, Matrix Eigensystem Routines-EISPACK Guide Extension, Springer-Verlag, 1977.

[4] Kung, H.T., "Systolic Arrays (for VLSI)", In [illegible] and G.W. Stewart, Sparse Matrix Proceedings, 1978, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, [illegible].

[5] [illegible]

[6A] [illegible]

[6B] [illegible]

[7] [illegible] Applications", [illegible]

[8] Griffiths, L.J., "A Continuously-Adaptive Filter Implemented as a Lattice Structure", 1977 IEEE International Conference on Acoustics, Speech and Signal Processing Record, IEEE Catalog No. 77CH1197 - 3, ASSP pp. 683 - 686.

[9] Monzingo, R.A. and T.W. Miller, <u>Introduction to Adaptive Arrays</u>, Wiley, 1980, Ch. 6: "Direct Inversion of the Sample Covariance Matrix".

[10] Haykin, S. (Editor), <u>Nonlinear Methods of Spectral Analysis</u>, Springer-Verlag, 1979.

SIGNAL PROCESSING WITH STATE-SPACE MODELS[1]

Bernard C. Levy

Laboratory for Information and Decision Systems

Department of Electrical Engineering and Computer Science

Massachusetts Institute of Technology

Cambridge, MA 02139

ABSTRACT

This paper presents an overview of fast signal processing algorithms based on state-space models. The basic method for estimating signal with state-space models is to use a Kalman filter. However, the Kalman filter is not always well conditioned numerically, and to guarantee good numerical properties it must be implemented in square root form. We will therefore discuss state-space estimation algorithms from the square root point of view. In this context, it will be shown how this stationarity or close to stationarity of the signal process to be estimated can be exploited. This gives rise to the so called Chandrasekhar recursions. A scattering framework for linear estimation will also be in produced: and we will show that this framework can be used to obtain doubling algorithms, and to obtain parallel or decentralized implementations of signal processing algorithms. These implementations are based on a segmentation of the data, either in time, or in terms of information, and they provide the starting point for the VLSI implementation of state- space estimation algorithms with parallel processors.

[1]This work was supported by the National Science Foundation under Grant ECS - 80 - 12668 and by the Air Force Office of Scientific Research under Grant No. AFOSR - 82-0135.

A HIERARCHICAL SET OF VLSI LAYOUT GENERATORS FOR SIGNAL PROCESSING

M. Buric
Bell Laboratories
Murray Hill, NJ

(Abstract not available for print)

A Thousand-Word Speech-Recognition System Using Special-Purpose MOS-LSI

Hy Murveit
Meni Lowy
Bob Brodersen
Electronics Research Laboratory
UC Berkeley

Abstract

We have designed a MOS-LSI speech recognition system that is capable of accurately recognizing a large vocabulary of isolated words in real time. This system uses two custom-designed integrated circuits, memory, and a control microprocessor. We have built a TTL prototype of the system, evaluated its performance and found that it recognizes speech very accurately.

1. The Speech-Recognition Problem

Our system is similar to other dynamic-time-warped automatic speech recognition (ASR) systems [1,2,3]. It maintains a dictionary of model words or *templates* to which all input words are compared. The template that is most similar to the word just spoken is recognized and some associated action is performed. If no words are similar enough to the input, no action is performed. This dictionary can either be filled by the user in a training phase prior to usage (speaker-dependent mode) or it can be fixed in read-only memories and used in a speaker-independent mode following techniques suggested by Rabiner et al. [3].

Our MOS-LSI-based architecture for the system is shown in the figure. The "front end" integrated circuit analyzes [illegible] input speech, decides when a word was spoken, and passes on an internal representation of the word to a word comparator. Each comparator can process up to 1000 vocabulary words (500 seconds of speech) in real time. The microprocessor collects the outputs of the word comparators and uses them along with other syntactic or semantic information to make its recognition decision. The microprocessor then passes on the recognized word to the host system or performs some further application system task.

Our system can also be configured to recognize connected speech. In this mode it uses the isolated-word recognizer to spot words inside the phrase and then it ties these words together using an algorithm performed by the control microprocessor.

2. The Speech-Recognition Algorithm

A word in our ASR system is represented as a series of spectra (or *frames*), each representing 10ms or more of input speech. When sound enters our system, it is presented to a bank of bandpass filters, full-wave rectified, and lowpass-filtered. The resulting values are the average inband amplitude of the speech. Thay are sampled every 10ms and sent to a word endpoint detection system. Here the frames representing speech input are separated from background. This decision is based upon the energy level of the input and some knowledge of the nature of speech (minimum lengths of words, maximum amounts of silence inside words, etc.). The frames are then amplitude normalized (so that speech intensity does not affect recognition accuracy) and converted to four-bit logarithmic format. Frames are then selectively transmitted[1] to the comparator IC. After this downsampling our effective frame sampling rate is reduced to about 20 milliseconds.

The comparator IC matches all words in its dictionary with the input and finds a comparison error for each of these vocabulary items. It does this by summing the squared euclidean distances between corresponding frames of

[1]We only transmit frames whose euclidean distance to the previously transmitted frame are above a given threshold.

the input and the template being searched. However, local variations in speaking rate make it difficult to find these corresponding frames accurately (i.e., to time align the input and the template). This word alignment, one of the most computationally intensive parts of the system, is done by the following dynamic-programming-based recursion. Here $W_{in}(t)$ and $W_{tp}(t)$ are the frames of the input word and a template word, L_{in} and L_{tp} are the lengths of the two words, $d()$ is the frame distance measure and $D_{Lin,Ltp}$ is the error in comparing the two words.

$$D^{tp}_{i,j} = d(W_{in}(i),\ W_{tp}(j)) + MIN \left\{ D^{tp}_{i-1,j},\ D^{tp}_{i,j-1},\ D^{tp}_{i-1,j-1} \right\} \tag{1}$$

$$d(W_{in}(j), W_{tp}(k)) = \sum_{i=1}^{P} (W_{in}(j,i) - W_{tp}(k,i))^2 \tag{2}$$

We compute $D^{tp}_{i,j}$ for $i=1 \rightarrow L_{tp}$ and for $j=1 \rightarrow L_{in}$. We initialize $D^{tp}_{0,0}=0$ and $D^{tp}_{0,j}=D^{tp}_{i,0}=\infty\ \ i,j>0$. P is the number of filters (12).

To recognize speech in real time we require that the word comparator compute $D^{tp}_{i,j}\ i=1 \rightarrow L_{tp}$ for each template, every 20ms. Thus to time align an input word with each entry in a 1000 word dictionary and assuming words of average length 0.5 sec (25 frames) during 20ms., we would have to compute equation (2) at the frequency of:

$$f_{distance} = \frac{(1000\ words)(25\ frames/words)}{20msec.}$$

or 1.25 million times per second.

A critical aspect of the system is the size of the memory needed for storing the reference vocabulary and the size of the temporary memory. To store 500 seconds of speech represented every 20 msec. by twelve 4-bit words, we need 300,000 4 bits or 1,200,000 bits of memory. We are evaluating a vector-quantization scheme that would reduce this to one sixth or one eigth of its original value [7].

3. Evaluations

We have built a TTL prototype speech recognition system that uses the same algorithms and has the same word length restrictions as our IC design. It can recognize a thousand words of isolated speech in real time. We are currently adding connected speech and speaker independence capabilities to it. This system has helped us obtain much information about the human-factors aspects of the speech-recognition problem, and in this way allowed us to improve parts of our ASR system. This system has been interfaced to a graphics editor for integrated circuits and used by several people in our laboratory. We also use this prototype system to evaluate the performance of our IC-based ASR system.

To compare our system to existing speech recognizers, we obtained a set of audio tapes from George Doddington and Tom Schalk of Texas Instruments. These tapes were used by Doddington and Shalk at TI to evaluate commercial speech recognizers[4]. We ran these tapes once through our ASR system. The following table shows the performance of our system inserted into a table taken from Doddington and Shalk s study.

Manufacturer	**Model**	**Nominal price**	**errors**
Verbex	1800	$65,000	10 (0.2%)
UC Berkeley			45 (0.9%)
Nippon Electric	DP-100	$65,000	60 (1.2%)
Threshold Technology	T-500	$12,000	73 (1.4%)
Interstate Electronics	VRM	$2,400	147 (2.9%)
Heuristics	7000	$3,300	300 (5.9%)
Centigram	MIKE 4725	$3,500	366 (7.1%)
Scott Instruments	VET/1	$500	646 (12.6%)

REFERENCES

[1] G.M. White, "Speech Recognition Experiments with Linear Prediction, Bandpass Filtering and Dynamic Programming", *IEEE Trans. Acoust.,Speech,Signal Processing*, Vol. ASSP-24,pp. 183-188, Apr. 1976

[2] L.R. Rabiner and S.E. Levinson, "Isolated and Connected Word Recogniton - Theory and Selected Applications", *IEEE Trans. Comm.*, Vol. COM-29,pp. 621-659, May 1981

[3] L.R. Rabiner J.G. Wilpon, "Considerations in Applying Clustering Techniques to Speaker-independent Word Recognition", Vol. 66-3,pp. 663-673, Sept. 1979

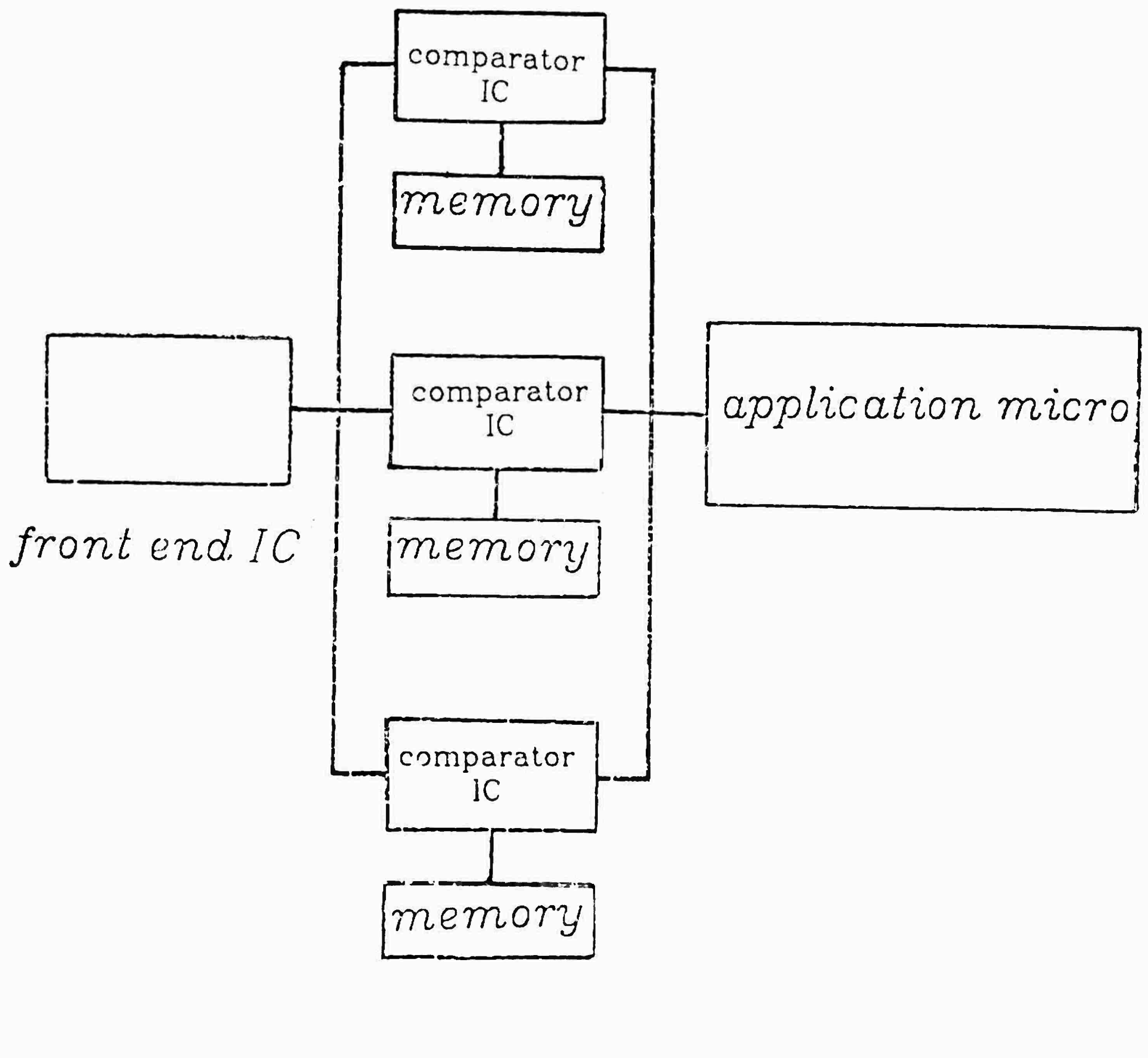
[4] G.R. Doddington and T.B. Schalk, "Speech Recognition: Turning Theory to Practice", *IEEE Spectrum*, pp. 26-32, Sept. 1981

[5] H. Murveit, *Speech Recognition by Computer*, Ph.D. Thesis, University of California, Berkeley,to be published.

[6] M. Lowy,*Implementation of a Speech Recognition System*. Ph.D. Thesis, University of California, Berkeley,to be published.

[7] R. Kavaler,*Vector Quantization of Spectra for a Speech Recognition System*, M.S. Thesis, University of California, Berkeley,to be published.

Fig. 1. Block Diagram of the Speech Recognition System.



A PARALLEL VLSI ARCHITECTURE FOR A DIGITAL FILTER USING A NUMBER THEORECTIC TRANSFORM

T. K. Truong

Communication System Research
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91103

I. S. Reed, C.-S. Yeh, H. M. Shao

Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089

This work was supported in part by the U.S. Air Force Office of Scientific Research, under grant number AFOSR-80-0151, in part by JPL Director's Discretionary Fund, FY82 under contract number 730-00231-0-3310, in part by the contract N00039-80-C-0641, Dept. of Navy, Naval Electronics Systems Command, and also in part by NIH contract No. 1R01 GM28473-01.

Let $F_t = 2^{2^t} + 1$ be the t-th Fermat number for $t \geq 0$. Let $\{x_n\}$ be an N-point sequence of integers, where $0 \leq x_n \leq F_t - 1$, $0 \leq n \leq N-1$, and N is a power of 2. The Fermat number transform pair of $\{x_n\}$ and $\{X_k\}$ over F_t is defined as follows:

$$X_k \equiv \sum_{n=0}^{N-1} x_n \alpha^{nk} \pmod{F_t}, \quad k=0,1,\ldots,N-1 \tag{1}$$

$$x_n \equiv (1/N) \sum_{k=0}^{N-1} X_k \alpha^{-nk} \pmod{F_t}, \quad n=0,1,\ldots,N-1 \tag{2}$$

where $0 \leq X_k \leq F_t - 1$ and α is an N-th root of unity. More details of an FNT can be found in [1,2].

In this paper, F_t, α, and N are selected specifically to be $F_5 = 2^{32}+1$, 2, and 64, respectively. The dynamic range [3] of this FNT is $(2^{32}/(2 \cdot 64))^{1/2} = \sqrt{2} \cdot 12^{12}$, which is sufficiently large for a number of applications. Eqs. (1) and (2) have a mathematical algorithm similar to the FFT. Hence the pipeline structure shown in Fig. 1 can be used to compute a 64-point FNT [4]. The symbolic diagram and operations of a FNT butterfly are shown in Fig. 2.

In the following, the overlap-save method is generalized to implement a digital filter. Let $\{h_m\}$ be the filter sequence of a digital filter, where $0 \leq m \leq M-1$. Suppose $\{x_n\}$ and $\{y_k\}$ are input data and output data sequences of the filter, respectively, where $0 \leq n \leq N-1$ and $0 \leq k \leq N+M-1$ [4]. For purposes of exposition it is assumed in the following argument that M=96 and N=512.

In order to use 64-point FNT's to compute $\{y_k\}$, three 64-point subfilters $\{h_m^1\}$, $\{h_m^2\}$, and $\{h_m^3\}$ are formed by partitioning $\{h_m\}$ as follows:

$$h_m^i = \begin{cases} h_{m+32(i-1)} & \text{for } 0 \le m \le 31 \\ 0 & \text{for } 32 \le m \le 63 \end{cases} \tag{3}$$

for $1 \le i \le 3$. Next the overlap-save method [4] is used to compute the linear convolution $\{y_k^i\}$ of $\{x_n\}$ and $\{h_m^i\}$ by using the cyclic convolution technique, where $1 \le i \le 3$ and $0 \le k \le 543$. To accomplish this $\{x_n\}$ is sectioned into 64-point subsequences with 32 points of $\{x_n\}$ overlapped between two consecutive subsequences. That is $\{x_n\}$ is sectioned into $\{x_m^1\} = \{x_m\}$, $\{x_m^2\} = \{x_{m+32}\}, \ldots,$ $\{x_m^{15}\} = \{x_{m+448}\}$, where $0 \le n \le 511$ and $0 \le m \le 63$. Then $\{y_k^i\}$, for $1 \le i \le 3$, is computed by overlapping the cyclic convolution of $\{h_m^i\}$ and $\{x_m^j\}$ for $1 \le j \le 15$ using 64-point FNT's. Finally the output sequence $\{y_k\}$, for $0 \le k \le 512+96-1 = 607$, results evidently from $\{y_k^k\}$ for $1 \le i \le 3$ by the following equation:

$$y_k = y_k^1 + y_k^2 z^{-32} + y_k^3 z^{-64} \tag{4}$$

In Fig. 3 is shown the block diagram of an architecture for the generalized overlap-save method of a digital filter using one FNT and three inverse FNT's of 64 points.

The advantages of this architecture for implementing a digital filter using FNT transforms are the following: (1) It requires no multiplications. Only additions and bit rotations are needed. (2) It alleviates the usual dynamic range limitation for long sequence FNT's. (3) It utilizes the FNT and inverse FNT circuits 100% of the time. (4) The lengths of the input data and filter sequences can be arbitrary and different. (5) It is regular, simple, and expandable, and as a consequence suitable for VLSI implementation.

REFERENCES

1. R. C. Agarwal and C. S. Burrns, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-22, No. 2, pp. 87-97, April 1974.

2. R. C. Agarwal and C. S. Burrns, "Number Theoretic Transforms to Implement Fast Digital Convolution," Proc. IEEE, Vol. 63, No. 4, pp. 550-560, April 1975.

3. I. S. Reed and T. K. Truong, "The Use of Finite Field to Compute Convolutions," IEEE Trans. Information Theory, Vol. IT-21, No. 2, pp. 208-213, March 1975.

4. L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Inc., Inglewood Cliffs, New Jersey, 1975.

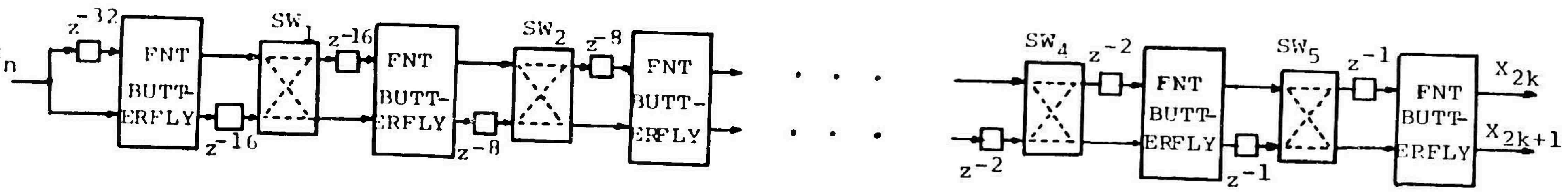
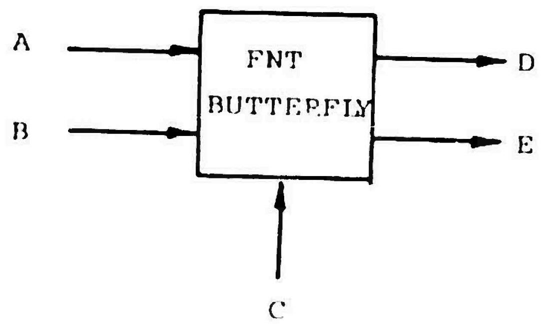


Figure 1. A Pipeline Structure for Computing a 64-point Fermat Number Transform.



(a)

$D \leftarrow A + B\cdot(2)^C$

$E \leftarrow A - B\cdot(2)^C$

Figure 2. (a) The symbolic diagram and
(b) Operations of an FNT butterfly.

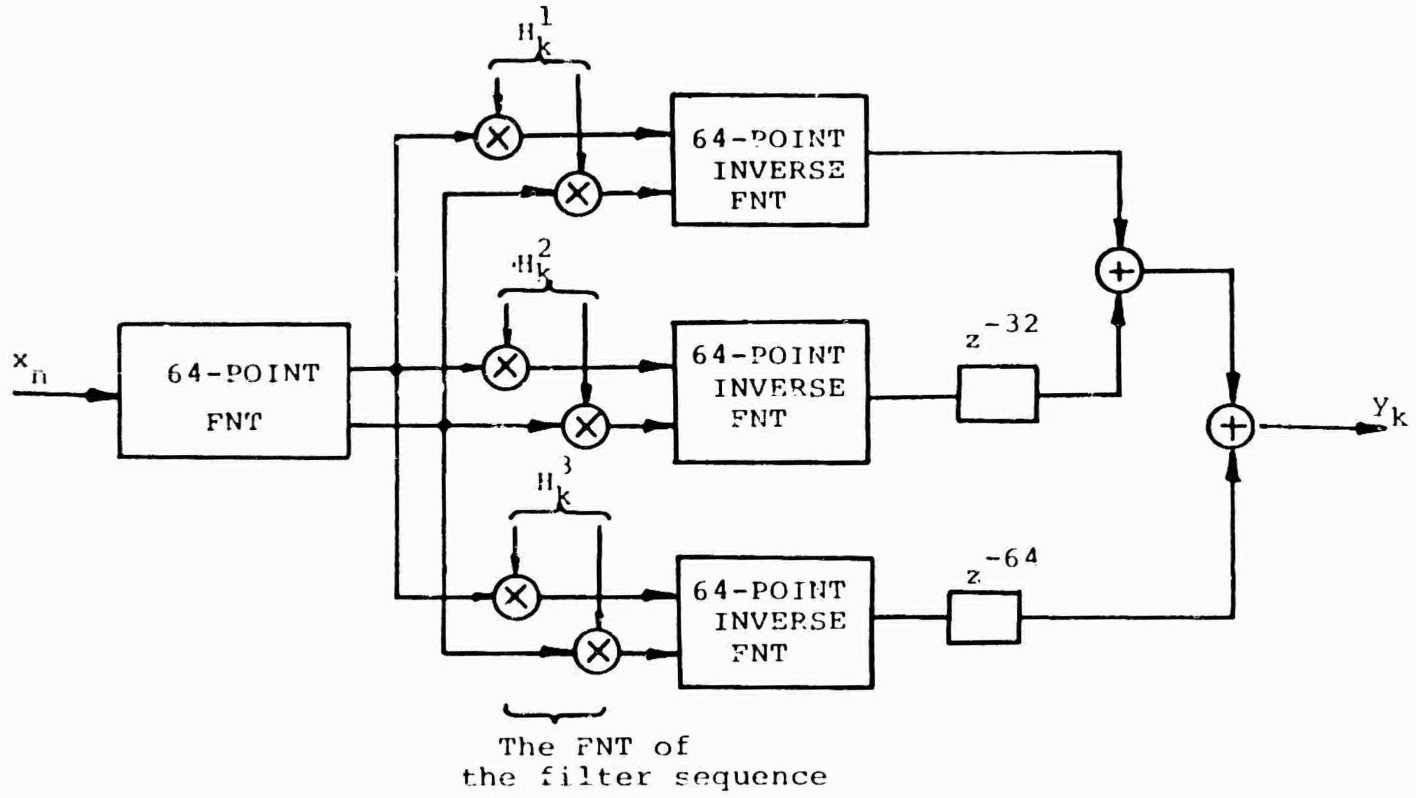


Figure 3. A Realization of a Digital Filter with a Filter Sequence of 96 points by using the generalized overlap-save method and the FNT technique.

FREQUENCY DOMAIN DIGITAL FILTERING WITH VLSI

Earl E. Swartzlander, Jr.
TRW Defense Systems Group
One Space Park
Redondo Beach, CA 90278

George H. Hallnor
TRW Defense Systems Group
7600 Colshire Dr.
McLean, VA 22102

INTRODUCTION

During the last decade, a new generation of integrated circuits has been developed that is directly applicable to the implementation of advanced digital signal processors. Examples of such circuits include microprocessors, fast wide word memories, single chip multipliers, floating point adders, etc. Although of independent interest, such circuits are significant primarily as components for the development of more complex systems. This paper shows how one such system, a very high performance digital filter, is implemented using current technology.

In particular, a processor with throughput on the order of one billion radix 2 butterflies per second is shown to be feasible. Such high speed performance may be required, for example, by advanced digital signal processing systems such as adaptive beam formers. Processor designs such as that described here should provide guidance for future VLSI device research.

The selected system requirements typify demanding applications. They involve filtering data at input rates of 40 Mega-Samples Per Sec. (MSPS) with a resolution of 10 KHz. This requires computation of the spectrum of 4096 complex input data, multiplication of the spectrum by a selected filter response, and inverse transformation to produce the filtered time domain signal. This processing flow is repeated for data skewed by 50% of the transform length to eliminate edge effects due to data windowing.

The design approach is to convert the input 40 MSPS data stream into four 10 MSPS data streams. These four streams are processed using a Radix 4 Cooley-Tukey FFT pipeline processor according to the basic design of McClellan and Purdy of the MIT Lincoln Laboratory [Ref 1]. The Cooley-Tukey FFT algorithm was selected because both the forward and inverse transforms are computed using the same set of basic computational elements. The McClellan and Purdy design was developed nearly a decade ago for implementation using small scale ECL circuits by General Electric Company for use at Kwajelein Missile Range. The GE implementation uses a room full of equipment (tens of thousands of integrated circuits) to perform a 16 K point FFT with an input data rate of 120 MSPS [Ref. 1]. The current design is thus slower than the predecessor but vastly simpler.

FREQUENCY DOMAIN FILTER REQUIREMENTS

The basic algorithm flow is shown in Figure 1. Input data flows (along the upper path) through a data acquisition element/buffer into a 4192 point complex FFT. Data windowing is performed in the time domain in the data acquisition element. The transformed data is filtered by multiplication in the frequency domain followed by inverse FFT processing. The magnitude of the unfiltered spectrum is computed to determine the power spectral density which is used to develop the filter kernel. A similar process is performed on data skewed in arrival time (by half of the transform length) through the lower (overlap) channel.

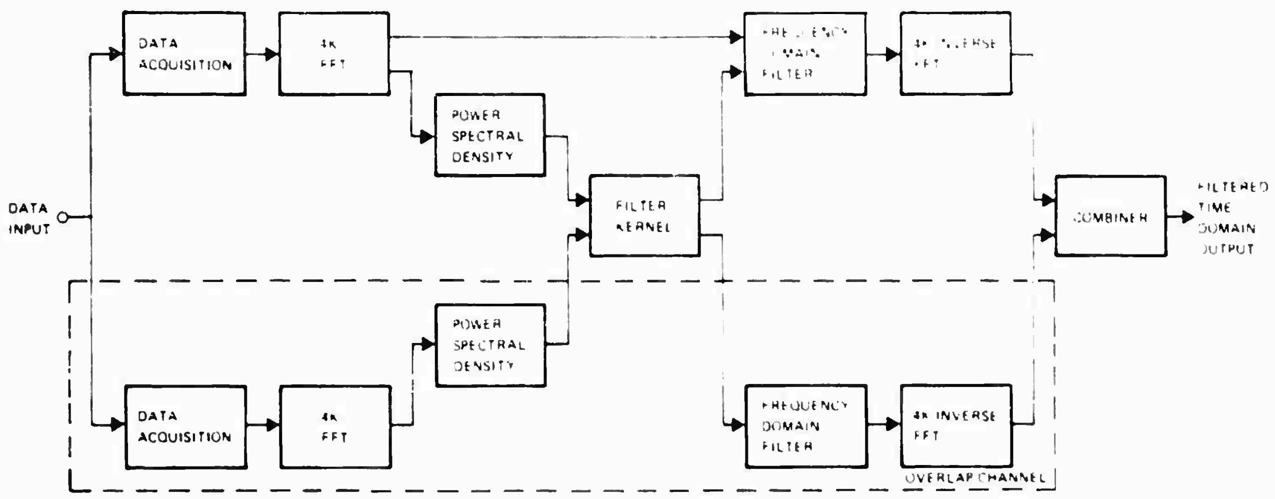


Figure 1. FFT Frequency Domain Filter

Examination of this algorithm flow indicates the need for the following modules.

- Data Acquisition: provides input (data source) interface, rate conversion from a single 40 MSPS channel to four 10 MSPS channels, and windowing with any prestored time domain window (see [Ref. 2] for comparison of windows).

- 4192 Point Complex FFT: accepts data in time sequential order and produces the complex FFT. The transform element is programmable to accommodate transforms of various binary lengths (i.e., 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc.).

- Frequency Domain Filter: multiplies the FFT spectrum by a spectral filter. This is a simple point by point multiplication.

- Power Spectral Density: computes the magnitude of the spectrum, orders the data in frequency order, and averages using an exponential weight.

- Filter Kernel: a general purpose computer develops an adaptive filter kernel based on the observed power spectral density.

- 4192 Point Complex Inverse FFT: converts the filtered spectrum back to the time domain. The inverse FFT is programmable to accommodate transforms of various binary lengths.

Thus six major modules are required. It should be emphasized that these modules are not of the same level of complexity; the FFT and inverse FFT modules are much more complex than the other modules.

After the initial processing in the data acquisition module, all computation is implemented with 22-bit floating point arithmetic (6-bit exponent, 16-bit fraction) to maintain precision and avoid the need for adaptive data scaling.

FFT MODULE DESIGN

Since the FFT module is the most complex portion of the system, examination of its implementation will demonstrate some of the system level issues. Due to its complexity the FFT module is partitioned into functional elements of two types. This partitioning is a compelling advantage of the Cooley-Tukey pipeline FFT algorithm. For the FFT module four parallel channels are implemented using pipeline processing with a cascade architecture [Ref. 3]. This choice was made to achieve high speed (to accommodate 40 MSPS data rates) while using readily available VLSI technology (with current devices 10 MHz clock rates are easily achieved, higher speeds would require specialized circuits and much lower levels of functional integration). A wide variety of VLSI components are currently available in several technologies to support 10 MHz clock rates. Examples include high speed 16K bit static Random Access Memory (RAM), 22-bit floating point adders and multipliers [Ref. 4], etc.

The structure of the FFT processor is shown in Figure 2. It is constructed with Computational Elements (CE) and Delay/Commutator (D/C) Elements. In this design the computational element is a Radix 4 butterfly and the Delay/Commutator element performs data reordering required for the Cooley-Tukey FFT algorithm [Ref. 5]. The basic pipeline processing concept was developed for Radix-2 FFT implementation by Raytheon in the late 1960s [Ref. 6], and is currently being implemented in VLSI by NEC [Ref. 7].

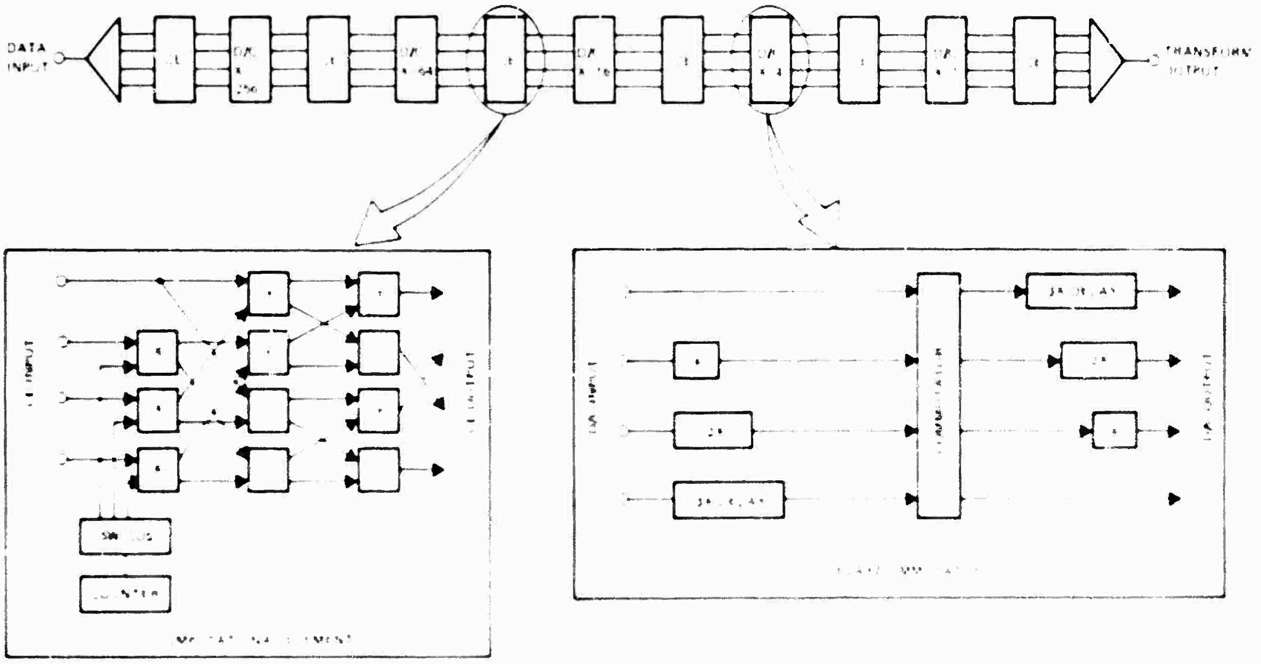


Figure 2. Radix 4 Pipeline Cooley-Tukey FFT

The Radix 4 butterfly computational element realizes a 4 point discrete Fourier transform. It is implemented with 3 complex multipliers, 3 complex adders (4 perform addition and 4 perform subtraction), Sin-Cos tables (stored in ROM), and miscellaneous addressing logic. The computational element is implemented with approximately 80 integrated circuits. This low level of complexity is a direct result of the availability of single chip arithmetic components.

The other element of the pipeline FFT is the delay/commutator which is used to perform the interstage data reordering required by the Cooley-Tukey algorithm. It consists of input delay lines of graduated length, a commutator switch, and output delay lines of graduated length. The length of the unit delay is 1 at the final stage of the FFT and increases by a factor of 4 at each preceding stage. The module requires about 180 integrated circuits to implement transform lengths up to 4096 points. The delays are implemented with a RAM into which data is written in sequence through the memory and the delayed data is read in sequence from the memory.

COMPLETE FREQUENCY DOMAIN FILTER

The total parts complement for the filter processor 52 circuit cards containing approximately 6300 integrated circuits and dissipating about 6 kW as detailed in Table 1. The modules are packaged on circuit cards measuring 9 by 16 inches. This construction allows the use of commercial wirewrap cards and cages. The circuit cards are packaged in a standard equipment rack. Each of the four transform processors (i.e., forward, overlapped forward, inverse, and overlapped inverse) require a drawer.

Table 1. Fast Transform Processor Complexity

Module Type	Module Count	Total Chips	Total Power
Data Acquisition & Weighting	2	240	269
FFT			
Computational Element	12	960	1536
Delay/Commutator	10	1790	990
Frequency Domain Filter	2	140	213
Power Spectral Density	2	78	154
Combiner	2	312	312
Inverse FFT (Same as FFT)	22	2750	2526
Total	52	6270	[illegible]

CONCLUSION

This paper illustrates the potential for achieving throughputs of 1 billion butterflies per second with VLSI technology. Table 2 summarizes the characteristics of the complete filter. Two significant figures of merit are the computational rate per unit volume (960 million butterflies per sec in 13.2 cu ft for 72 million butterflies per sec/cu ft) and the computational rate per watt (approximatelly 160 thousand butterflies per sec/watt). These figures of merit and the computational rate may be viewed in perspective by noting that the SPS-1000 with 4 pipes achieves about 42 million butterflies per sec [Ref. 8]. The computational rate per unit volume is on the order of 50-100 million butterflies per sec/cu ft. This study certainly does not indicate an upper bound on processor throughput. Indeed, it is reasonable to speculate that at least computational rates on the order of 100 billion butterfly operations per sec may be achieved with advanced ECL or GaAs technology in the near future.

Table 2. Frequency Domain Filter Characteristics

Technology:	off-the-shelf 1982 Logic : Bipolar Memory: NMOS
Size :	13.2 cu ft (using wirewrap construction)
Power :	6 kW
Throughput:	960 Million Radix-2 Butterflies per sec.
Word Size :	Input : 16 bit fixed point Intermediate: 22 bit floating point (6 bit exponent, 16 bit fraction)
Figures of Merit:	72 Million butterflies per sec/cu ft 160,000 butterflies per sec/watt

REFERENCES

1. J. H. McClellan and R. J. Purdy, "Applications of Digital Signal Processsing to Radar" in A. V. Oppenheim, Editor, <u>Applications of Digital Signal Processing</u>, Prentice-Hall, Englewood Cliffs, 1978, Ch. 5.

2. F. J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform," <u>Proceedings of the IEEE</u>, Vol. 66, 1978, pp. 51-83.

3. G. D. Bergland, "Fast Fourier Transform Hardware Implementations - An Overview," <u>IEEE Transactions on Audio and Electroacoustics</u>, Vol. AU-17, 1969, pp. 104-109.

4. J. A. Eldon and C. Robertson, "A Floating Point Format for Signal Processing," <u>Proceedings ICASSP-82</u>, 1982, pp. 717-720.

5. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, Vol. 19, 1965, pp. 297-301.

6. H. L. Groginsky and G. A. Works, "A Pipeline Fast Fourier Transform," IEEE Transactions on Computers, Vol. C-19, 1970, pp. 1015-1019.

7. A. Kanemasa, R. Maruta, K. Nakayama, Y. Sakamura, and S. Tanaka, "An LSI Chip Set for DSP Hardware Implementation," Proceedings ICASSP-81, 1981, pp. 644-647.

8. R. A. Collesidis, T. A. Dutton, and J. R. Fisher, "An Ultra-High Speed FFT Processor," Proceedings ICASSP-80, 1980, pp. 784-787.

Concurrent Algorithms as Space-time Recursion Equations*

Marina C. Chen and Carver A. Mead[‡]

Introduction

It is by now well recognized that VLSI technology has brought about a medium which allows the realization of orders of magnitude more computing elements per unit cost. The more significant contribution of VLSI to Computer Science will be in the utilization of many hundreds or thousands of these elements concurrently to achieve a given computation. It is clear from existence proofs of such innovative designs as systolic arrays [KUNG & LEISERSON80], tree machine algorithms [BROWNING80], computational arrays [JOHNSSON ET AL.81], wavefront arrays [KUNG,S.Y.80], etc. that vast performance improvements can be achieved if the design of so-called "high-level" algorithms is released from the one dimensional world of a sequential process, and the cost of communications in space as well as cost of computation in time is taken into consideration [SUTHERLAND & MEAD77]. While this higher dimensional design space provides a great playground for innovative algorithm design, it also introduces pitfalls unapprehended by those accustomed to the world of a single sequential process. Verification of algorithms becomes much more crucial in system designs because debugging concurrent programs can very easily become an exponentially complicated task in this rich space. The real difficulty lies in the high degree of complexity of concurrent systems. The well-known hierarchical approach can be used to manage the design complexity for such systems. A system is broken down into successive levels of sub-systems until each is of a manageable complexity. The effectiveness of this approach relies on two basic tools: A design and verification methodology for each level and an abstraction mechanism to go from one level to the next. The latter is crucially important, for without it the consistency of the whole system is imperiled.

In this paper, we describe a methodology and a single notation for the specification and verification of synchronous and self-timed concurrent systems ranging from the level of transistors to communicating processes. The uniform treatment of these systems results in a powerful abstraction mechanism which allows management of system complexity.

Traditionally, due to the assumption that the cost of accessing variables in memory is the same regardless of their locations, sequential algorithms ignore the spatial relationships

*This work is sponsored by System Development Foundation and its initial phase by Defense Advanced Research Projects Agency ARPA Order #3771 and monitored by Office of Naval Research Contract #N00014-79-C-0567. One of us (M.C.) was supported by an IBM Doctoral Fellowship

[‡]Computer Science Department, California Institute of Technology, Pasadena, California.

of variables. In addition, the steps of a computation have not been explicitly expressed as a function of time, but are rather implied by programming constructs. Languages that cannot express the spatial relationships of variables cannot take into account the most important aspect in the design of a concurrent algorithm, i.e. ensuring locality of communications, taking advantage of the interplay of variables in space (in practice up to 3 dimensions) to achieve higher performance. The implicit "time" causes programming languages to suffer either from not being able to abstract the history of computation (e.g. in applicative and data-flow languages [KAHN74, BAKUS78]), or not being able to abstract computation in a clean functional form (e.g. in assignment-based languages). Here we choose to make "time" an explicit parameter of computation. We call our representation of computation a "Space-time Algorithm".

In [CHEN82], CRYSTAL (Concurrent Representation of Your Space Time ALgorithm), a notation for concurrent programming is proposed. The fixed-point approach [SCOTT & STRACHEY71] is used for characterizing the semantics. Within this framework, a program is expressed as a set of systems of recursion equations. Unknowns of the equations are data expressed as functions from the space-time domain to the value domain. For a deterministic concurrent system, such as a systolic array, a single system of equations results, and the semantics of such a system is defined as the least solution of the equations. The semantics of concurrent systems in general can be characterized as the corresponding set of solutions of the set of systems of equations. In this paper, we concentrate on deterministic concurrent systems at the communicating sequential processes level. We will first present briefly considerations that are general to all systems, i.e., the underlying model of computation, the representation, and the mathematical semantics of the systems. Various inductive techniques (see for example [MANNA74]) used in verifying recursive programs can be directly applied in verifying space-time algorithms and proving their properties. We demonstrate this framework by presenting both the synchronous and self-timed version of the matrix multiplication on systolic arrays [Kung & Leiserson80] with its proof of correctness. The notion of wavefront is especially important in this class of computations. We define the "phase" of a computation wave in a way that is analogous to the wave in physical world. The set of all possible "phases" can be formalized as a well-founded set, upon which the inductive proof is based.

Model of Computation

The model consists of an ensemble of *sequential processes* each of which has its own local state and ports for communicating with other processes. Depending on the level of system concerned, these processes can be as simple as a single transistor or as fancy as a conventional von Neumann type machine. A sequential process consists of a function that maps from inputs and current-states to outputs and next-states. Such a function uniquely defines a sequential process. It is the generator of the output sequence and state with given initial state and an input sequence. The state captures the semantic abstraction of the history. No assertion about the process can depend upon history in a way not captured by the state. A single invocation of the function i) evaluates the function, ii)updates the state and outputs, iii)increments the process's "time", all as an atomic event.

In a particular process, "time" is a measure of how many invocations have occurred, and "space" is where the process is located. "Time" is a property local to each process. Note that state is explicitly represented, a function is not defined from the history of inputs to outputs as in the applicative and data flow model of computation. Communications among processes in space are specified by identifying inputs of one process with outputs of other processes in the space-time domain. Also note that a transition from one invocation to the next within a sequential process can be viewed as a communication in the time domain (fixed in space).

The "slicing" of a sequential process into a sequence of functions is done at the communication with the external world. Inputs from several different processes which are aligned in time and used as arguments to a single function are considered as one external input event, i.e. one invocation. Within the same slice, no side-effects are allowed, i.e. each slice is strictly functional. We enforce this discipline by using a purely applicative programming notation (like pure Lisp and Backus's FP notation) to implement atomic functions, which cannot be further sliced either in space or in time. Any higher level system is constructed by composing atomic functions and other existing systems using recursion equations. The resulting system is always transformed into a function from inputs and current-states to outputs and next-states, i.e. a sequential process. It is often the case that once the system is implemented, a sequence of inputs can be conveniently considered as a set of inputs at the next level up. Although internal state is used as part of the implementation, the outputs can be expressed as a function of such a sequence of inputs without refering to the state. In such a case we abstract the process as an applicative function and it can, once again, be treated as if it were atomic. In real-time systems, this kind of abstraction is not possible since the sequence of inputs cannot be treated as a static input, making explicit state still necessary.

Thus space-time algorithms are either purely applicative programs or recursion equations. Note that in this way, states can be expressed without side-effects. The change from viewing an applicative system as the universe to using it only for an atom is the key to the applicability of our framework to real systems. The applicative model of computing suffers one major drawback in not being able to retain the result of a computation so that it can be used in a different place or at a later time. The data flow model is a remedy for this problem only in space. The essential ability to use a result in several places is captured by the data-flow equations devised by Kahn [KAHN74]. Unfortunately, this model still lacks the essential capability of capturing the state, the result in the time domain. This fact is manifested in the proliferation of assignment-based data-flow languages [ACKERMAN, W.B.]. The elegance of data-flow equations cannot help the implementation of real world systems where state is necessary. The space-time recursion equations using a purely applicative language can be applied to real world problems*.

*The drawback of applicative languages is best captured in a quote by Perlis "Purely applicative languages are poorly applicable" [PERLIS 82]

This insight is the most important contribution of our work to computer programming. We thus retain the elegance and formal cleanness of functional application together with the essential ability to abstract history into a compact form.

Representation

Let $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ denote the inputs and current-states of a process, where $x_i \in \mathcal{D}_i$ for $i = 1, 2, \ldots, m$. Here $\mathcal{D}_i$ is the domain* where the input or current-state i assumes its value. We define a function f which maps the vector of inputs and current states to the vector of outputs and next states:

$$f : \mathcal{D}^m \rightarrow \mathcal{D}^n,$$
$$f = (\lambda\mathbf{x}.f_1, \lambda\mathbf{x}.f_2, \ldots, \lambda\mathbf{x}.f_n) \qquad (1)$$

where n is the total number of outputs and states.

Each component, $\lambda\mathbf{x}.f_i$, of such a function is an element of $[\mathcal{D}^m \rightarrow \mathcal{D}]$. These functions must be monotonic (see for example [MANNA 74]) over $\mathcal{D}^m$.

In order to capture the notion of flow of the data and the structure of these data, we define *data streams*. Each "stream" of data is represented by a function from the space-time domain $\mathcal{V}$ to the value domain $\mathcal{D}$. We next define *structured processes* which define the location of the processes making up the ensemble, as a function in the domain $[\mathcal{V} \rightarrow [\mathcal{D}^m \rightarrow \mathcal{D}]]$. This function is defined by cases for different process types in the space. Cases are specified in the notation of [DIJKSTRA 76]. The complexity of this definition reflects the heterogeneity of the process types.

The relationship among the structured input and output data streams and structured processes and functions are defined in a point-wise manner. In the space-time domain, an output is the result of the application of the function at that point to input data streams at that point. Through connections, the input (current-state) streams of one function are identified with output (next-state) streams of other functions, or with initial/boundary values. We therefore define *structured connections* also as a function in the domain $[\mathcal{V} \rightarrow [\mathcal{D}^m \rightarrow \mathcal{D}]]$. The description of this function reflects the regularity of the connections. By substituting the equations of structured connections into those of structured processes, we obtain a system of recursion equations that define output streams in terms of output streams and initial/boundary conditions.

An obvious restriction on these recursion relations is that the time components can only increase by one unit at a time, i.e. an argument presented to the input of a function at its "time" t will affect the value of that function which appears at its output at its "time" $t + 1$.

*Technically, domains are not sets but *complete lattices* with *approximation ordering*. Please refer to [MANNA 74] and [SCOTT & STRACHEY 71].

In general, an input stream at a given point in the space-time domain can connect to an output stream at any other point in space. In specific cases, such as when the low-cost neighboring communications are used, inputs are connected to outputs of neighboring processes. In the case of such neighboring connections, the relations are local in all dimensions. It is then possible to use difference equations for our specification. Recursion relations retain more information in the sense that the "phase" of a computation wave is embedded in the description. For more complex situations, involving non-local connections, the greater generality of recursion relations is essential.

Semantics and Abstraction

By the well-known fixed-point theory[LASSEZ, NGUYEN &SONENBERG 82], the unique minimum solution of any system of recursion equations exists. This minimum solution is taken to be the function that the system computes. The process of finding this minimum solution can be described intuitively by the following successive approximation procedure. We first approximate the solution by the set of n data streams that are totally undefined in the space-time domain and substitute them into the right-hand side of the recursion equations. This substitution results in the left-hand side which is a set of data streams that are defined only on the point in space-time domain where initial values are set. These data streams are the inputs to the algorithm and we refer to them as initial streams. Now we substitute these initial streams again into the recursion equation and get another set of data streams that have even more points in the space-time domain defined. We repeat this process until no more points in the space-time domain become defined. This process corresponds exactly to the process of computation. The only restriction is that our functions must be monotonic, i.e. each data stream at any iteration is always at least as well defined as it was on the previous one. We do not allow non-monotonic functions that destroy results which have already become defined. Refer to [MANNA74] [STOY77] for the formalism.

The resulting minimum solution consists of n data streams. Each data stream is a function over $[\mathcal{V} \rightarrow \mathcal{D}]$. In order to construct a higher level system using the system we have just obtained, we need to encapsulate the system as a sequential process or a function. The function defining a sequential process or the single applicative function is from value domain to value domain. This encapsulation usually involves some transformation from the original data structure to the space-time domain for the inputs and a corresponding transformation for the outputs. Technically, the procedure is as follows:

(1) The input mapping function maps all initial/boundary values from an abstract input data structure to the inputs unconnected to any output in the space-time domain.

(2) Compute the least fixed point solution in the space-time domain as described above.

(3) The resulting outputs occur at those points in the space-time domain designated as outputs of the system. The output mapping function maps these outputs from an element of $[\mathcal{V} \rightarrow \mathcal{D}]^n$ to an element of the output value domain $\mathcal{D}^{n'}$.

The result of this procedure is the abstract definition in the value domain, of the function (type $[\mathcal{D}^{m'} \rightarrow \mathcal{D}^{n'}]$) implemented by the space-time algorithm.

Matrix Multiplication on a Systolic Array—Program and Semantics

In his paper, Kung described various matrix related operations performed on an array of interconnected hexagonal elements. We present his algorithm for multiplying two full matrices in CRYSTAL and prove the correctness of the algorithm.

As shown in figure 1, hexagonal elements are connected into a hexagonal array. Each element has three inputs and three outputs as shown by the incoming and outgoing arrows, respectively. Such a process performs an inner product operation in the north and south direction , i.e. $c_{out} = c_{in} + a_{in} \times b_{in}$, and transmits the other two inputs as they were., i.e., $a_{out} = a_{in}, b_{out} = b_{in}$.

The two matrices to be multiplied, A and B, and a matrix C are fed into the array as shown in the figure. The resulting matrix C' will come out at the top of the array as shown. Kung's original algorithm assumes a global clock thus every process performs an operation synchronously. When data items are fed from the boundaries of the array, due to the fact that every process is forced to perform an operation even before any meaningful data reaches the process, proper initialization of the system by padding zeros in the input streams and disposal of garbage data are necessary. The same algorithm with a different timing scheme, e.g. self-timed [Seitz 80] scheme can simplify conceptually the interaction of processes and the flow of data and renders a simpler initiation of the system. This simplification results from the fact that the self-timed scheme assures that each process does not perform any operation until all the meaningful data items have reached the process. On the other hand, the self-timed scheme does not have any global control, the ordering of the system events is an emergent property of the local synchronizations. Thus the specification of the ordering relations among invocations of processes has to be verified from initial data arragement since self-timed elements are triggered by the arrival of data.

Both algorithms can be described by CRYSTAL programs. We will present both versions as examples of our notation and verification methodology and discuss some of the design issues of synchronous systems vs. self-timed systems.

In writing a CRYSTAL program, one need to choose an appropriate coordinate system for the processes in space. The data flow of the array has a symmetry which can be described by the dihedral group of order 3 [LIN & MEAD 82]. As shown in Figure 1, the 3-dimensional Cartesian coordinate system is chosen to reflect this symmetry. The center

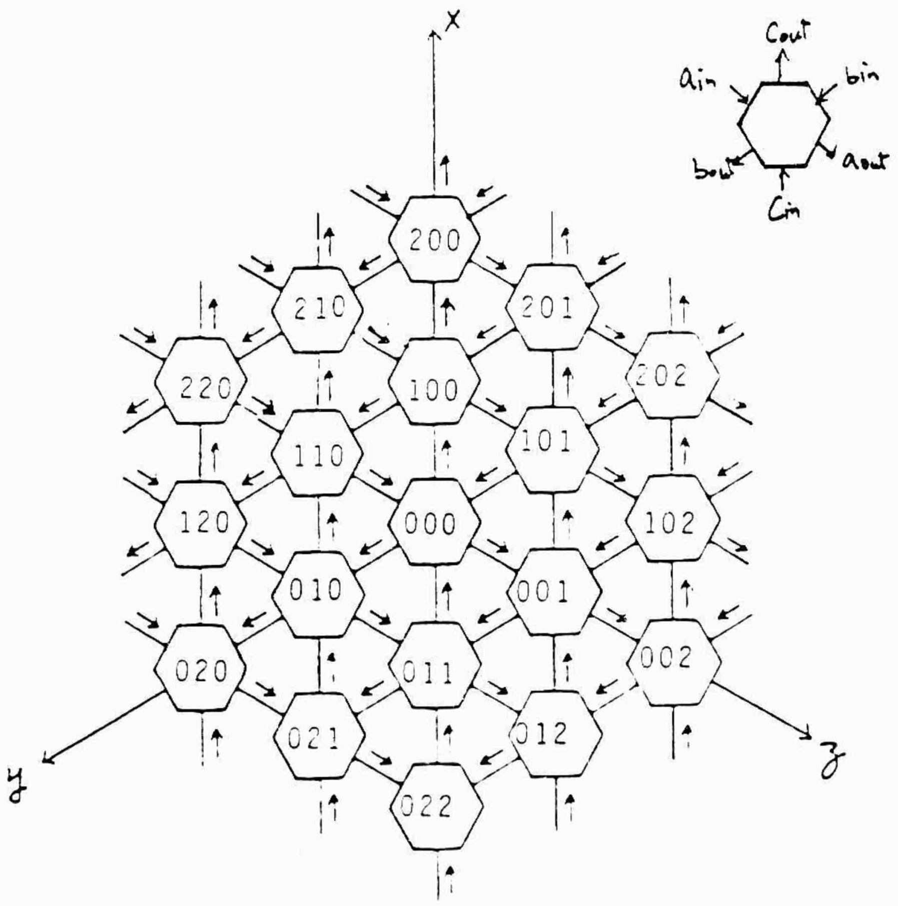


FIGURE 1. The Space Coordinate System For A Hexagonal Array

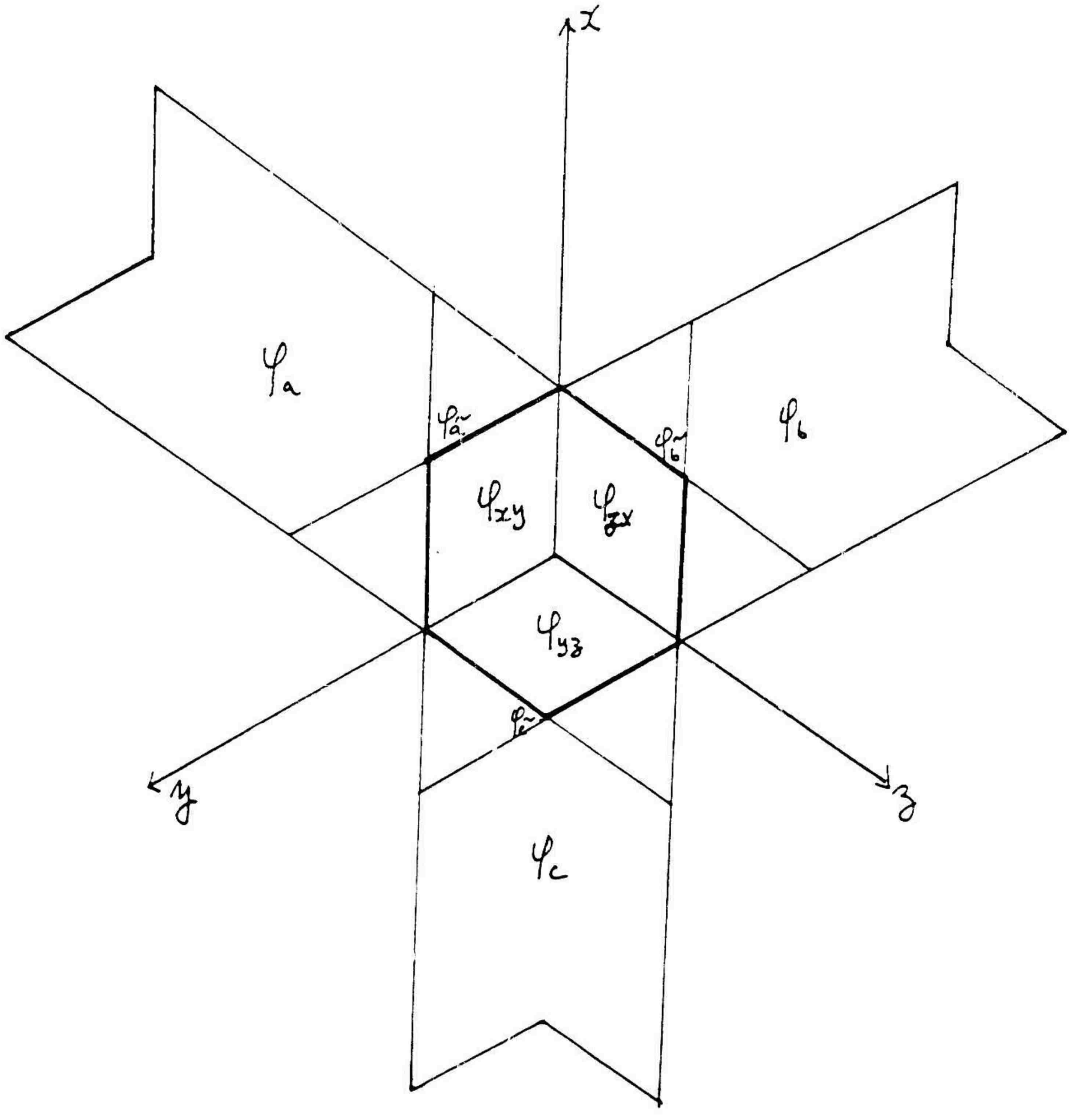

x
y
z
φ_a
φ_b
φ_c
$\varphi_{\tilde a}$
$\varphi_{\tilde b}$
$\varphi_{\tilde c}$
φ_{xy}
φ_{zx}
φ_{yz}

FIGURE 2

of this hexagon can be viewed as a corner of a cube. The hexagon is made of the three faces that contain the corner.

Next, we choose a coordinate system in the time domain according to the system timing scheme. If the system is synchronous, then t denotes the number of system clock cycles. For a self-timed system, each process has its own time frame. If it is a deterministic system, there exists a unique partial ordering of all events of the system. For a nondeterministic system, there exists more than one possible partial ordering of system events. Thus the synchronous system is a special case of deterministic systems where the unique partial ordering on events is controlled by the system clock.

Let x, y, z, t be non-negative integers. Define the following predicates which specify the location of processes in the space-time domain as shown in Figure 2. For example φ_{xy} restricts the xy plane to an area within the specified bounds in the first quadrant.

$$\begin{aligned}
\varphi_{xy} \equiv & (n > x > 0) \wedge (n > y > 0) \wedge (z = 0) \\
\varphi_{yz} \equiv & (n > y > 0) \wedge (n > z > 0) \wedge (x = 0) \\
\varphi_{zx} \equiv & (n > z > 0) \wedge (n > x > 0) \wedge (y = 0) \\
\varphi_{x} \equiv & (n > x > 0) \wedge (y = 0) \wedge (z = 0) \\
\varphi_{y} \equiv & (n > y > 0) \wedge (z = 0) \wedge (x = 0) \\
\varphi_{z} \equiv & (n > z > 0) \wedge (x = 0) \wedge (y = 0) \\
\varphi_{xyz} \equiv & (x = 0) \wedge (y = 0) \wedge (z = 0) \\
\varphi_{h} \equiv & \varphi_{xy} \vee \varphi_{yz} \vee \varphi_{zx} \vee \varphi_{x} \vee \varphi_{y} \vee \varphi_{z} \vee \varphi_{xyz} \\
\varphi_{a} \equiv & (0 \leq |x - y| < 3n) \wedge (0 \leq \min(2x - y, 2y - x) < 3n) \wedge (z = 0) \\
\varphi_{b} \equiv & (0 \leq |z - x| < 3n) \wedge (0 \leq \min(2z - x, 2x - z) < 3n) \wedge (y = 0) \\
\varphi_{c} \equiv & (0 \leq |y - z| < 3n) \wedge (0 \leq \min(2y - z, 2z - y) < 3n) \wedge (x = 0) \\
\varphi_{s} \equiv & \varphi_{a} \vee \varphi_{b} \vee \varphi_{c} \\
\varphi_{a'} \equiv & \varphi_{xy} \vee \varphi_{zx} \vee \varphi_{x} \\
\varphi_{b'} \equiv & \varphi_{yz} \vee \varphi_{xy} \vee \varphi_{y} \\
\varphi_{c'} \equiv & \varphi_{xy} \vee \varphi_{zx} \vee \varphi_{x} \\
\varphi_{t} \equiv & 0 \leq t \leq 4(n - 1) + 1
\end{aligned}$$

We use the notation $\overline{\varphi}$ to indicate the negation of the predicate φ. Define the space-time domain of the array $V \equiv \{(x, y, z, t) : \varphi_{s} \wedge \varphi_{t}\}$. From now on unless otherwise specified, (x, y, z, t) always refers to any $(x, y, z, t) \in V$.

Let A_{in}, B_{in}, and C_{in} be the input data streams (a function over $[V \rightarrow D]$) and A_{out}, B_{out} and C_{out} be the output data streams. Then the following process definition specify how each output stream is related to the input streams. For example, each hexagonal element within the hexagon (when φ_h holds), has an inner product element for computing

c_{out} and two delay elements for computing a_{out} and b_{out}. The definition below defines for all the elements in the space-time domain in a structured way.

Process Definition

$$A_{out} = \lambda(x, y, z, t).\begin{cases} \varphi_a \vee \varphi_{a'} \rightarrow A_{in}(x, y, z, t-1) \\ else \rightarrow \perp \end{cases} \tag{2a}$$

$$B_{out} = \lambda(x, y, z, t).\begin{cases} \varphi_b \vee \varphi_{b'} \rightarrow B_{in}(x, y, z, t-1) \\ else \rightarrow \perp \end{cases} \tag{2b}$$

$$C_{out} = \lambda(x, y, z, t).\begin{cases} \varphi_c \wedge \overline{(\varphi_{yz} \vee \varphi_y \vee \varphi_z)} \rightarrow C_{in}(x, y, z, t-1) \\ \varphi_h \rightarrow C_{in}(x, y, z, t-1) + A_{in}(x, y, z, t-1) \times B_{in}(x, y, z, t-1) \\ else \rightarrow \perp \end{cases} \tag{2c}$$

Next we define the connection plans for all the elements. It specifies which output connects to which input in a structured way.

Connections

$$A_{in} = \lambda(x, y, z, t).\begin{cases} t = 0 \rightarrow A_{in}(x, y, z, t) \\ t > 0 \rightarrow \begin{cases} \varphi_a \rightarrow A_{out}(x+1, y+1, z, t) \\ \varphi_{a'} \rightarrow A_{out}(x, y, z-1, t) \end{cases} \\ else \rightarrow \perp. \end{cases} \tag{3a}$$

$$B_{in} = \lambda(x, y, z, t).\begin{cases} t = 0 \rightarrow B_{in}(x, y, z, t) \\ t > 0 \rightarrow \begin{cases} \varphi_b \rightarrow B_{out}(x+1, y, z+1, t) \\ \varphi_{b'} \rightarrow B_{out}(x, y-1, z, t) \end{cases} \\ else \rightarrow \perp. \end{cases} \tag{3b}$$

$$C_{in} = \lambda(x, y, z, t).\begin{cases} t = 0 \rightarrow C_{in}(x, y, z, t) \\ t > 0 \rightarrow \begin{cases} \varphi_c \rightarrow C_{out}(x, y+1, z+1, t) \\ \varphi_{c'} \rightarrow C_{out}(x-1, y, z, t) \end{cases} \\ else \rightarrow \perp. \end{cases} \tag{3c}$$

By substituting (3a) into (2a), we obtain

$$A_{out} = \lambda(x, y, z, t).\begin{cases} t = 1 \rightarrow \begin{cases} \varphi_a \rightarrow A_{in}(x, y, z, 0) \\ \varphi_{a'} \rightarrow 0 \end{cases} \\ t > 1 \rightarrow \begin{cases} \varphi_a \rightarrow A_{out}(x-1, y-1, z, t-1) \\ \varphi_{a'} \rightarrow A_{out}(x, y, z, t-1) \end{cases} \end{cases} \tag{4}$$

Similarly, we can substitute (3b) into (2b) and (3a), (3b), (3c) into (2c) to obtain a system of recursion equations in A_{out}, B_{out} and C_{out} and the initial conditions.

These equations define the behavior of the hexagonal array itself independent of the input to the array. Next, we specify the input and output transformation functions which relate the structure of the hexagonal array with the abstract data structure of matrices.

Let $h^1 = (h_a^1, h_b^1, h_c^1)$ denote the initial streams and $h^\infty = (h_a^\infty, h_b^\infty, h_c^\infty)$ denote the final streams which are the minimum solution of the above system of recursion equations (4). A Matrix with elements from the domain $\mathcal{D}$ can be thought of as a function from the domain $\mathcal{N}^2$ of index pairs to $\mathcal{D}$. We denote the resulting matrix by A', B' and C' and define the following domain,

$$\begin{aligned} \text{domain of integers from 0 to } n-1\text{:}\ & \mathcal{N} \\ \text{domain of matrices:}\ & \mathcal{M} \equiv [\mathcal{N}^2 \rightarrow \mathcal{D}], \\ \text{domain of data streams:}\ & \mathcal{S} \equiv [\mathcal{V} \rightarrow \mathcal{D}], \\ \text{domain of transformation functions:}\ & \mathcal{T} \equiv [\mathcal{V} \rightarrow \mathcal{N}^2] \\ & \mathcal{T}' \equiv [\mathcal{N}^2 \rightarrow \mathcal{V}] \end{aligned} \tag{5}$$

We define the input transformation function $(g_a, g_b, g_c) \in \mathcal{T}^3$:

$$g_a \equiv (I_a, J_a), \quad g_b \equiv (I_b, J_b), \quad g_c \equiv (I_c, J_c)$$

where

$$I_a \equiv \lambda(x,y,z,t).\begin{cases} 2y - x \equiv 0 \pmod 3 \rightarrow \frac{2y-x}{3} \\ else \rightarrow \perp \end{cases}$$

$$J_a \equiv \lambda(x,y,z,t).\begin{cases} 2z - y \equiv 0 \pmod 3 \rightarrow \frac{2z-y}{3} \\ else \rightarrow \perp \end{cases} \tag{6a}$$

$$I_b \equiv \lambda(x,y,z,t).\begin{cases} 2x - z \equiv 0 \pmod 3 \rightarrow \frac{2x-z}{3} \\ else \rightarrow \perp \end{cases}$$

$$J_b \equiv \lambda(x,y,z,t).\begin{cases} 2z - x \equiv 0 \pmod 3 \rightarrow \frac{2z-x}{3} \\ else \rightarrow \perp \end{cases} \tag{6b}$$

$$I_c \equiv \lambda(x,y,z,t).\begin{cases} 2y - z \equiv 0 \pmod 3 \rightarrow \frac{2y-z}{3} \\ else \rightarrow \perp \end{cases}$$

$$J_c \equiv \lambda(x,y,z,t).\begin{cases} 2z - y \equiv 0 \pmod 3 \rightarrow \frac{2z-y}{3} \\ else \rightarrow \perp \end{cases} \tag{6c}$$

We use the shorthand notation

$$g_a(x,y,z,t) \equiv (I_a(x,y,z,t), J_a(x,y,z,t))$$

for component-wise application of the argument (x, y, z, t) to a vector of functions such as (I_a, J_a).

then the inital streams are defined as

$$h_a^1 = \lambda(x,y,z,t). \begin{cases} t = 0 \rightarrow \begin{cases} \varphi_a \rightarrow \begin{cases} (2y - x \equiv 0 \ (\mathrm{mcd}\ 3)) \wedge (2x - y \equiv 0 \ (\mathrm{mod}\ 3)) \\ \qquad \rightarrow Ag_a(x,y,z,t) \\ else \rightarrow 0 \end{cases} \\ \varphi_{a'} \rightarrow 0 \\ else \rightarrow \perp \end{cases} \\ t > 0 \rightarrow \perp \end{cases}$$

$$h_b^1 = \lambda(x,y,z,t). \begin{cases} t = 0 \rightarrow \begin{cases} \varphi_b \rightarrow \begin{cases} (2x - z \equiv 0 \ (\mathrm{mod}\ 3)) \wedge (2z - x \equiv 0 \ (\mathrm{mod}\ 3)) \\ \qquad \rightarrow Bg_b(x,y,z,t) \\ else \rightarrow 0 \end{cases} \\ \varphi_{b'} \rightarrow 0 \\ else \rightarrow \perp \end{cases} \\ t > 0 \rightarrow \perp \end{cases} \tag{7}$$

$$h_c^1 = \lambda(x,y,z,t). \begin{cases} t = 0 \rightarrow \begin{cases} \varphi_c \rightarrow \begin{cases} (2y - z \equiv 0 \ (\mathrm{mod}\ 3)) \wedge (2z - y \equiv 0 \ (\mathrm{mod}\ 3)) \\ \qquad \rightarrow Cg_c(x,y,z,t) \\ else \rightarrow 0 \end{cases} \\ \varphi_{c'} \rightarrow 0 \\ else \rightarrow \perp \end{cases} \\ t > 0 \rightarrow \perp \end{cases}$$

$$\text{where } h_a^1, h_b^1, h_c^1 \in S, \qquad A, B, C \in \mathcal{M}, (^*in)$$

Output Transformation Function $(g'_a, g'_b, g'_c) \in \mathcal{T}'^3$:

$$g'_a \equiv (X_a, Y_a, Z_a, T_a), \quad g'_b \equiv (X_b, Y_b, Z_b, T_b), \quad g'_c \equiv (X_c, Y_c, Z_c, T_c).$$

where

$$\begin{aligned} X_a &\equiv \lambda(i,j).\max(j - i, 0), \\ Y_a &\equiv \lambda(i,j).\max(i - j, 0), \\ Z_a &\equiv \lambda(i,j).n - 1. \\ X_b &\equiv \lambda(i,j).\max(i - j, 0), \\ Y_b &\equiv \lambda(i,j).n - 1, \\ Z_b &\equiv \lambda(i,j).\max(j - i, 0). \\ X_c &\equiv \lambda(i,j).n - 1, \\ Y_c &\equiv \lambda(i,j).\max(i - j, 0), \\ Z_c &\equiv \lambda(i,j).\max(j - i, 0). \\ T_a \equiv T_b \equiv T_c &\equiv \lambda(i,j).n - \min(i,j) - i - j. \end{aligned} \tag{8}$$

As before, $g'_a(i,j)$ denotes the component-wise application of (i,j) to the four components of g'_a.

The resulting matrices are defined as follows,

$$A' = h_a^{\infty} g'_a, B' = h_b^{\infty} g'_b, C' = h_c^{\infty} g'_c,$$

where $h_a^{\infty}, h_b^{\infty}, h_c^{\infty} \in S$, the final streams

$$A', B', C' \in \mathcal{M}; \qquad g'_a, g'_b, g'_c \in \mathcal{T}'(^*out)$$

We now verify that the above system of recursion equations and the input output transformation functions correctly implement the familiar matrix operations, i.e.

$$A'(i,j) = A(i,j) \tag{9a}$$

$$B'(i,j) = B(i,j) \tag{9b}$$

$$C'(i,j) = \sum_{k=0}^{n-1} A(i,k) \times B(k,j) + C(i,j) \tag{9c}$$

$$\text{where } 0 \leq i < n, \quad 0 \leq j < n$$

We verify first the following lemmas which are the final streams (the solution of the recursion equations) in the space-time domain.

Lemma A,B:

$$A_{out}(x,y,z,t) = \begin{cases} \varphi_a \vee \varphi_{a'} \rightarrow \\ \quad A_{in}(x + \max(t-1-z,0), y + \max(t-1-z,0), \max(z-(t-1),0), 0) \\ else \rightarrow \perp \end{cases} \tag{10a}$$

$$B_{out}(x,y,z,t) = \begin{cases} \varphi_b \vee \varphi_{b'} \rightarrow \\ \quad B_{in}(x + \max(t-1-y,0), \max(y-(t-1),0), z + \max(t-1-y), 0) \\ else \rightarrow \perp \end{cases} \tag{10b}$$

Lemma C:
Let $U_1 \equiv t-1+k-y$, $U_2 \equiv t-1+k-z$, $V_1 \equiv (t-1-x-k)-(y+k)$, $V_2 \equiv (t-1-x-k)-(z+k)$, $K_1 \equiv 1-\min(x,t-1)$ and $K_2 \equiv \min(n-1-y, n-1-z, t-1-x)$. Define

$$S_1 \equiv \sum_{k=K_1}^{0} A_{in}(x+k+\max(U_2,0), y+\max(U_2,0), \max(-U_2,0), 0)$$
$$\times B_{in}(x+k+\max(U_1,0), \max(-U_1,0), z+\max(U_1,0), 0)$$

and

$$S_2 \equiv \sum_{k=0}^{K_2} A_{in}(\max(V_2,0), y+k+\max(V_2,0), \max(-V_2,0), 0)$$
$$\times B_{in}(\max(V_1,0), \max(-V_1,0), z+k+\max(V_1,0), 0),$$

then

$$C_{out}(x,y,z,t) = \begin{cases} \varphi_c \vee \varphi_{c'} \rightarrow \\ \quad C_{in}(\max(x-(t-1),0), y+\max(t-1-x,0), z+\max(t-1-x,0), 0) \\ \qquad +\lambda(x,y,z,t).S_1(x,y,z,t)+\lambda(x,y,z,t).S_2(x,y,z,t) \\ else \rightarrow \perp \end{cases} \tag{10c}$$

These relate outputs on any point in space-time domain to the initial input streams. Since they are total functions in space-time domain, the solution of the equations is automatically the minimum. Thus one way of verify them is simply substitute them into the recursion equation and check if the equations hold. The simple substitution technique will not work in general, since the final streams are not necessarily total in the space-time domain. In this case, an inductive proof showing that the final streams are the minimum solution is necessary.

The computation waves of such a system are very instructive in such proofs. We observe that there are two triangular waves, one incident wave proceeding toward the origin. Another is a reflected wave proceeding outward from the origin. We define the phase of the reflected wave to be $x+y+z-t$ and that of the incident wave to be $x+y+z+2t$. A wave front is defined in the traditional way as the locus of all points of the wave having the same phase. With t fixed, there are many of such wavefronts spread out in space. In this particular system, we number these wavefront positions by $w = x+y+z$. Notice that the partial result of a particular element of a matrix is carried on by a single wavefront with one value of phase. Intuitively, the induction for the reflected wave is on the wavefront of phase ϕ and $w=k$ to phase ϕ and $w=k+1$. For the incident wave, inductions proceeds from $w=k$ to $w=k-2$.

The pair (ϕ, w) can be formalized as a well-founded set (a set with no infinite decreasing sequences, so that induction is valid on the set) with the binary relation $\prec$, which is defined as the transitive closure of $\prec^*$, the binary relation defined below only on neighboring elements:

$$\begin{aligned} &\text{incident wave: } (\phi_1,w_1) \prec^* (\phi_2,w_2) \text{ if } \phi_1=\phi_2 \text{ and } w_1=w_2+2. \\ &\text{reflected wave: } (\phi_1,w_1) \prec^* (\phi_2,w_2) \text{ if } \phi_1=\phi_2 \text{ and } w_1=w_2-1. \end{aligned} \tag{11}$$

The inductive proof is given in [CHEN 82].

By composing the input and output transformation functions with the initial and final streams, we obtain (9a), (9b), and (9c). The detailed proof is in [CHEN 82].

Self-timed Systems

A self-timed system differs from a synchronous system in the sequencing of system events. In a synchronous system, all processes are activated simultaneously by the same clock cycle, i.e. all processes have the same invocation number. The invocations of each process are ordered by the linear sequence of the clock. Thus all the invocations (s,t), where s is the space parameter and t is the time parameter, have a unique partial ordering $\prec$ defined to be

$$(s_1,t_1) \prec (s_2,t_2) \text{ if } s_1 = s_2 \text{ and } t_1 < t_2.$$

In a self-timed system, the ordering of the system events is not self-evident as in the synchronous system. Each process is invoked only when all of its inputs are ready. Thus the overall system timing is an emergent property of the ensemble attributed by the local synchronization of all the processes in the system. In such a system, the "time" component of the space-time domain is a function of the space component, i.e., each process has its own time-frame. The relation between time-coordinates of two communicating processes needs to be asserted in the connection plans. This time-domain relationship among processes must be verified since it depends on the initial data arrangement because self-timed elements are triggered by the input data.

The following is the space-time algorithm for the self-timed matrix multiplication on the systolic array. We define a few more predicates to specify where the initial data will be put. Notice that this set of predicates covers much less area than the set φ_a, φ_b and φ_c, since the self-timed algorithm does not need padding zeros in the input data streams.

$$\begin{aligned}\varphi_{\bar{a}} \equiv& (0 \leq |x-y| < n) \wedge (0 \leq \min(x,y) < n) \wedge (z=0)\\ \varphi_{\bar{b}} \equiv& (0 \leq |z-x| < n) \wedge (0 \leq \min(z,x) < n) \wedge (y=0)\\ \varphi_{\bar{c}} \equiv& (0 \leq |y-z| < n) \wedge (0 \leq \min(y,z) < n) \wedge (z=0)\end{aligned}$$

We also redefine

$$\begin{aligned}\varphi_s \equiv& \varphi_{\bar{a}} \vee \varphi_{\bar{b}} \vee \varphi_{\bar{c}}\\ \varphi_t(x,y,z) \equiv& 0 \leq t \leq n-1-\max(x,y,z)\end{aligned}$$

Process Definition

$$A_{out} = \lambda(x,y,z,t).\begin{cases}\varphi_{\bar{a}} \vee \varphi_{a'} \to A_{in}(x,y,z,t(x,y,z))\\ else \to \perp\end{cases} \tag{12a}$$

$$B_{out} = \lambda(x,y,z,t).\begin{cases}\varphi_{\bar{b}} \vee \varphi_{b'} \to B_{in}(x,y,z,t(x,y,z))\\ else \to \perp\end{cases} \tag{12b}$$

$$C_{out} = \lambda(x,y,z,t).\begin{cases}\varphi_{\bar{c}} \wedge \overline{(\varphi_{yz} \vee \varphi_y \vee \varphi_z)} \to C_{in}(x,y,z,t(x,y,z))\\ \varphi_h \to C_{in}(x,y,z,t(x,y,z)) + A_{in}(x,y,z,t(x,y,z)) \times B_{in}(x,y,z,t(x,y,z))\\ else \to \perp\end{cases} \tag{12c}$$

Connection Plans

$$A_{in} = \lambda(x,y,z,t).\begin{cases}t=0 \to A_{in}(x,y,z,t(x,y,z))\\ t>0 \to \begin{cases}\varphi_{\bar{a}} \to A_{out}(x+1,y+1,z,t(x,y,z)-1)\\ \varphi_{a'} \to A_{out}(x,y,z-1,t(x,t,z))\end{cases}\\ else \to \perp.\end{cases} \tag{13a}$$

$$B_{in} = \lambda(x,y,z,t).\begin{cases} t=0 \rightarrow B_{in}(x,y,z,t(x,y,z)) \\ t>0 \rightarrow \begin{cases} \varphi_{\bar{b}} \rightarrow B_{out}(x+1,y,z+1,t(x,y,z)-1) \\ \varphi_{b} \rightarrow B_{out}(x,y-1,z,t(x,y,z)) \end{cases} \\ else \rightarrow \perp. \end{cases} \tag{13b}$$

$$C_{in} = \lambda(x,y,z,t).\begin{cases} t=0 \rightarrow C_{in}(x,y,z,t(x,y,z)) \\ t>0 \rightarrow \begin{cases} \varphi_{\bar{c}} \rightarrow C_{out}(x,y+1,z+1,t(x,y,z)-1) \\ \varphi_{c'} \rightarrow C_{out}(x-1,y,z,t(x,y,z)) \end{cases} \\ else \rightarrow \perp. \end{cases} \tag{13c}$$

Input transformation functions: The functions $(g_a, g_b, g_c) \in \mathcal{T}^3$ map from the space-time domain $\mathcal{V}$ to the matrix indices $\mathcal{N}^2$ as specified in (5).

$$g_a \equiv (\lambda(x,y,z,t).y, \lambda(x,y,z,t).x)$$

$$g_b \equiv (\lambda(x,y,z,t).x, \lambda(x,y,z,t).z)$$

$$g_c \equiv (\lambda(x,y,z,t).y, \lambda(x,y,z,t).z)$$

The initial streams are defined by using the composition of the input transformation functions and the matrix function.

$$h_a^1 = \lambda(x,y,z,t)\begin{cases} t=0 \rightarrow \begin{cases} \varphi_{\bar{a}} \rightarrow Ag_a(x,y,z,t) \\ else \rightarrow \perp \end{cases} \\ t>0 \rightarrow \perp \end{cases}$$

$$h_b^1 = \lambda(x,y,z,t)\begin{cases} t=0 \rightarrow \begin{cases} \varphi_{\bar{b}} \rightarrow Bg_b(x,y,z,t) \\ else \rightarrow \perp \end{cases} \\ t>0 \rightarrow \perp \end{cases}$$

$$h_c^1 = \lambda(x,y,z,t)\begin{cases} t=0 \rightarrow \begin{cases} \varphi_{\bar{c}} \rightarrow Cg_c(x,y,z,t) \\ else \rightarrow \perp \end{cases} \\ t>0 \rightarrow \perp \end{cases} \tag{14}$$

Output Transformation Functions: The functions $(g'_a, g'_b, g'_c) \in \mathcal{T}'^3$ define the space-time coordinates associated with each element of the output matrix.

$$g'_a \equiv (X_a, Y_a, Z_a, T_a), \quad g'_b \equiv (X_b, Y_b, Z_b, T_b), \quad g'_c \equiv (X_c, Y_c, Z_c, T_c).$$

where

$$\begin{aligned}
X_a &\equiv \lambda(i,j).j - \min(i,j),\\
Y_a &\equiv \lambda(i,j).i - \min(i,j),\\
Z_a &\equiv \lambda(i,j).n - 1 - \min(i,j).\\
X_b &\equiv \lambda(i,j).i - \min(i,j),\\
Y_b &\equiv \lambda(i,j).n - 1 - \min(i,j),\\
Z_b &\equiv \lambda(i,j).j - \min(i,j).\\
X_c &\equiv \lambda(i,j).n - 1 - \min(i,j),\\
Y_c &\equiv \lambda(i,j).i - \min(i,j),\\
Z_c &\equiv \lambda(i,j).j - \min(i,j).\\
T_a &\equiv T_b \equiv T_c \equiv \lambda(i,j).\min(i,j).
\end{aligned} \tag{15}$$

In the connection plan we have used the following assertions.

$$t(x,y,z) = \begin{cases} \varphi_{\bar{a}} \rightarrow t(x+1,y+1,z)+1 \\ \varphi_{a'} \rightarrow t(x,y,z-1) \\ \varphi_{\bar{b}} \rightarrow t(x+1,y,z+1)+1 \\ \varphi_{b'} \rightarrow t(x,y-1,z) \\ \varphi_{\bar{c}} \rightarrow t(x,y+1,z+1)+1 \\ \varphi_{c'} \rightarrow t(x-1,y,z) \end{cases} \tag{16}$$

Proof :

We prove these assertions by induction on the well-founded set of wavefront number k.

$$K = \{\, k = \frac{x+y+z}{3} + t(x,y,z) : x,y,z \text{ and } t \text{ are non-negative integers.} \}$$

with the usual less-than $(<)$ ordering on the rationals.

(i) base case: $k = 0$. Since x, y, z, t are all non-negative integers, we have $x = y = z = t = 0$. By (14), $A_{in}(0,0,0,0)$, $B_{in}(0,0,0,0)$ and $C_{in}(0,0,0,0)$ are initial data, thus the process is initiated. Notice also that none of the other processes can proceed since there is at least one input undefined for each of them. The induction hypothesis is vacuously true in this case.

(ii) induction step. We assume that the hypothesis (16) is true for all $k < k_0$. We then show that it must be true for $k = k_0$. The proof consists of three steps.

(1) For a given k, inputs to processes of invocation $t(x,y,z) - 1$ were generated by processes with $t(x,y,z) - 1$ or $t(x,y,z) - 2$. We use this fact later to demonstrate that the processes' invocations occur in lock step, i.e. no process invoked more frequently than any other.

(2) All outputs of the previous invocation have been taken before another invocation is initiated.

(3) All inputs to an invocation $t(x,y,z)$ come from invocations of $t(x,y,z)$ and $t(x,y,z)-1$, depending upon location.

We consider a process at (x,y,z) with $k_0 = \frac{x+y+z}{3} + t(x,y,z)$. From Figure 1, it has three inputs *a.in*, *b.in* and *c.i* from the following neighboring processes respectively.

$$a_{in} : \begin{cases} \varphi_{\bar{a}} \rightarrow (x+1, y+1, z) \\ \varphi_{a'} \rightarrow (x, y, z-1) \end{cases}$$

$$b_{in} : \begin{cases} \varphi_{\bar{b}} \rightarrow (x+1, y, z+1) \\ \varphi_{b'} \rightarrow (x, y-1, z) \end{cases}$$

$$c_{in} : \begin{cases} \varphi_{\bar{c}} \rightarrow (x, y+1, z+1) \\ \varphi_{c'} \rightarrow (x-1, y, z) \end{cases} \tag{17}$$

Since $k_0 - 1 < k_0$, the hypothesis is true for this same process at $t(x,y,z)-1$, the previous invocation. By the induction hypothesis (16), this invocation takes its inputs from the above processes at their time $t(x,y,z)-1$ or $t(x,y,z)-2$ depending on where the process is located.

This process provides outputs to the following neighboring processes.

$$a_{out} : \begin{cases} \varphi_{\bar{a}} \wedge (x>0) \wedge (y>0) \rightarrow (x-1, y-1, z) \\ \varphi_{a'} \vee \varphi_x \vee \varphi_y \rightarrow (x, y, z+1) \end{cases}$$

$$b_{out} : \begin{cases} \varphi_{\bar{b}} \wedge (x>0) \wedge (z>0) \rightarrow (x-1, y, z-1) \\ \varphi_{b'} \vee \varphi_x \vee \varphi_z \rightarrow (x, y+1, z) \end{cases}$$

$$c_{out} : \begin{cases} \varphi_{\bar{c}} \wedge (z>0) \wedge (y>0) \rightarrow (x, y-1, z-1) \\ \varphi_{c'} \vee \varphi_x \vee \varphi_y \rightarrow (x+1, y, z) \end{cases}$$

We assert that these outputs from the invocation number $t(x,y,z)-1$ of process (x,y,z) are taken by either invocation number $(t(x,y,z)-1)$ or $t(x,y,z)$ of these neighboring process depending upon their locations. Since

$$\frac{x+y+z+1}{3} + (t(x,y,z)-1) = \frac{x-1+y-1+z}{3} + (t(x,y,z)-1)+1 = k - \frac{2}{3} < k$$

Thus the induction hypothesis can be applied. Process (x,y,z) is ready to start a new invocation once all of its three inputs are ready. Since the processes that provide output to it at their respective time frame $t(x,y,z)-1$ or $t(x,y,z)$ satisfy the following inequality

$$\frac{x+1+y+1+z}{3} + t(x,y,z) - 1 = \frac{x+y+z-1}{3} + t(x,y,z) = k - \frac{1}{3} < k,$$

the induction hypothesis applies to them. Process (x, y, z) has three and only three inputs ready at their respective locations after its invocation number $t(x, y, z) - 1$. The align-element insures that no invocation can occur before all three inputs are ready, thus process (x, y, z) has its invocation number $t(x, y, z)$ occurring. This proves the above assertions.$\Box$

This algorithm is also deterministic for we can define a partial ordering $\prec$ on the invocations of processes. This binary relation is defined as the transitive closure of the binary relation $\prec^*$ as follows.

$$(x, y_1, z, t(x, y, z)) \prec^* (x_0, y_0, z_0, t(x_0, y_0, z_0))$$

If there exist $(x_1, y_1, z_1, t(x_1, y_1, z_1))$,$(x_2, y_2, z_2, t(x_2, y_2, z_2))$, and $(x_3, y_3, z_3, t(x_3, y_3, z_3))$ such that

$$\begin{cases} \varphi_{\bar{a}} \rightarrow ((x_1 = x_0) \wedge (y_1 = y_0) \wedge (z_1 = z_0 - 1) \wedge (t(x_1, y_1, z_1) = t(x_0, y_0, z_0))) \\ \varphi_{a'} \rightarrow ((x_1 = x_0 + 1) \wedge (y_1 = y_0 + 1) \wedge (z_1 = z_0) \wedge (t(x_1, y_1, z_1) = t(x_0, y_0, z_0) - 1)) \end{cases} \tag{18a}$$

and

$$\begin{cases} \varphi_{\bar{b}} \rightarrow ((x_2 = x_0) \wedge (y_2 = y_0 - 1) \wedge (z_2 = z_0) \wedge (t(x_2, y_2, z_2) = t(x_0, y_0, z_0))) \\ \varphi_{b'} \rightarrow ((x_2 = x_0 + 1) \wedge (y_2 = y_0) \wedge (z_2 = z_0 + 1) \wedge (t(x_2, y_2, z_2) = t(x_0, y_0, z_0) - 1)) \end{cases} \tag{18b}$$

and

$$\begin{cases} \varphi_{\bar{c}} \rightarrow (x_3 = x_0 - 1) \wedge (y_3 = y_0) \wedge (z_3 = z_0) \wedge (t(x_3, y_3, z_3) = t(x_0, y_0, z_0)) \\ \varphi_{c'} \rightarrow (x_3 = x_0) \wedge (y_3 = y_0 + 1) \wedge (z_3 = z_0 + 1) \wedge (t(x_3, y_3, z_3) = t(x_0, y_0, z_0) - 1) \end{cases} \tag{18c}$$

This definition can be derived from (16) with the existance of the align-element for the inputs of each process as an assumption.

Now we proceed to verify the algorithm by proving two lemmas.

Lemma a,b:

$$A_{out}(x, y, z, t) \equiv \begin{cases} \varphi_{\bar{a}} \vee \varphi_{a'} \rightarrow \\ \qquad A_{in}(x + t, y + t, 0, 0) \\ else \rightarrow \perp \end{cases} \tag{19a}$$

$$B_{out}(x, y, z, t) \equiv \begin{cases} \varphi_{b} \vee \varphi_{b'} \rightarrow \\ \qquad B_{in}(x + t, 0, z + t, 0) \\ else \rightarrow \perp \end{cases} \tag{19b}$$

Lemma C:

$$C_{out}(x, y, z, t) = \begin{cases} \varphi_{\bar{c}} \vee \varphi_{c'} \rightarrow C_{in}(\max(x - t, 0), y + \max(t - x, 0), z + \max(t - x, 0), 0) \\ \qquad + \sum_{k=0}^{x-t} A_{in}(k, y + t, 0, 0) \times B_{in}(k, 0, z + t, 0) \\ else \rightarrow \perp \end{cases} \tag{19c}$$

Similar to the synchronous case, we can either prove by direct substitution or by induction on K, the set of wavefront number. By composing these lemmas with input and output transformation functions, we can derive (9a), (9b) and (9c).

From both algorithms, we observe that the input and output transformation functions and the semantics of the hexagonal array are much simpler for the self-timed version. This result is not accidental, for the interaction among flows of data for this particular algorithm only utilizes one third of the maximum space-time resources. In the self-timed version, only one third of the processes (all processes with the same $k = \frac{x+y+z}{3} + t(x, y, z)$) are active at any instant. In the synchronous version, all processes are active at all times, thus padding zeros are necessary. The simplicity of the self-timed version is a pay-off of the more sophisticated synchronization method. It is necessary to prove that local synchronization gives rise to the global sequencing relations among all the processes. Describing these two algorithms in CRYSTAL not only shows the capability of our framework but also provides many insights into the the complexity of various aspects of these two different timing schemes. We have achieved one of the important goals of this research – by providing a formalism in which one can gain a much deeper understanding of the subject one describes in the process of so doing.

Conclusion

We have presented a notation and formal semantics for general non-linear systems with memory. An essential part of the semantics is a methodology for abstracting the behavior of such systems so they can be used as components at a higher level. The semantics of a particular system consists of

(i) An input mapping function from the value domain to the space-time structure of the system.

(ii) A function in the space-time domain which completely defines the operation of the system.

(iii) An output mapping function from the space-time structure to the value domain.

The abstract semantics of the system is obtained by eliminating space-time variables to yield a function in the value domain alone. When it is possible to eliminate all intermediate variables, as it was with the Kung array, the abstract system is purely functional. When some intermediate state variables remain, as in the case where real-time input is necessary, the system is defined by an abstract sequential process. Such a process is defined by a system of recursion relations in time. From an engineering point of view, the input and output mapping functions serve as precise interface specifications for the system.

The methodology can be applied to any system: linear, non-linear, time-varying, history-dependent. We believe it provides, for the first time, a unified approach spaning the range from computer programs to linear transfer functions; from transistor circuits to high level communicating sequential processes.

References

[ACKERMAN 82] Ackerman, W.B.
Data Flow Languages
Computer, 15(2):15-26, February 1982.

[BACKUS 78] Backus, J. *Can Programming Be Liberated from the von Neumann Style?*
A Functional Style and Its Algebra of Programs.
CACM, 21(8)613-641, August 1978.

[BROWNING 80] Browning, S.A.
The Tree Machine: A Highly Concurrent Computing Environment

Ph.D. thesis, California Institute of Technology, January, 1980.

[CHEN 82] Chen, M.C.
Ph.D.Thesis in Preparation, California Institute of Technology.

[DIJKSTRA 76] Dijkstra, E.W.
A Discipline of Programming.
Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[JOHNSSON 81] Johnsson, L., Weiser, U., Cohen, D. and Davis, A.
Towards a Formal Treatment of VLSI Arrays
Technical Report 4191, California Institute of Technology, January, 1981.

[KAHN 74] Kahn, G.
Proc. IFIP Congress 74.
The Semantics of a Simple Language for Parallel Programming,
1974.

[KUNG &LEISERSON 80]
Kung,H.T. and Leiserson C.E.
Algorithms for VLSI Processor Arrays.
Mead &Conway, Introduction to VLSI Systems Addison-Wesley, 1980, chapter 8.3.

[KUNG, S.Y. 80] Kung S. Y.
VLSI Array Processor for Signal Processing.

Conference on Advanced Research in Integrated Circuits, MIT., 1980.

[LASSEZ,NGUYEN&SONENBERG]
Lassez,J.L., Nguyen,V.L, and Sonenberg,E.A.
Fixed Point Theorems and Semantics: A Folk Tale
Information Processing Letters, 14(3):112-116, February 1982.

[LIN &MEAD] Lin T.Z. and Mead C.A.
The Application of Group Theory in Classifying Systolic Arrays
Display File 5006, California Institute of Technology, March, 1982.

[MANNA 74] Manna, Z.
Mathematical Theory of Computation.
McGraw-Hill, New York, 1974.

[PERLIS 82] Perlis, A.J.
Epigrams on Programming.
SIGPLAN Notices, 17(9), September 1982

[SEITZ 80] Seitz, C.
System Timing.
Mead &Conway, Introduction to VLSI Systems Addison-Wesley, 1980, chapter 8.3.

[SCOTT & STRACHEY 71]
Scott, D. and Strachey, C.
Toward a Mathematical Semantics for Computer Languages
Fox, J., editor. Polytechnic institute of Brooklyn Press, New York, 1971.

[STOY 77] Syoy, J.E.
Denotational Semantics: The Scott-Strachey Approach.
The MIT Press, Cambridge, Massachusetts, 1977.

[SUTHERLAND & MEAD 77]
Sutherland I. and Mead C.
Micro-electronics and Computer Science
Scientific American 237(3):210-229, September, 1977.

A FORMAL DERIVATION OF ARRAY IMPLEMENTATIONS OF FFT ALGORITHMS

Lennart Johnsson
Computer Science
California Institute of Technology
Pasadena, California 91125

Danny Cohen
Information Sciences Institute
University of Southern California
Marina Del Rey, California 90291

ABSTRACT

Fast Fourier Transform, FFT, algorithms are interesting for direct hardware implementation in VLSI. The description of FFT algorithms is typically made either in terms of graphs illustrating the dependency between different data elements or in terms of mathematical expressions without any notion of how the computations are implemented in space or time. Expressions in the notation used in this paper can be given an interpretation in the implementation domain. The notation is in this paper used to derive a description of array implementations of decimation-in-frequency and decimation-in-time FFT algorithms. Correctness of the implementations is guaranteed by way of derivation.

INTRODUCTION

The notation and the methodology used in this paper explicitly model storage, arithmetic, and logic operations. The notation has great resemblance with the notation used in the treatment of discrete-time systems and difference equations. The form used here was first described and given an interpretation in terms of computational networks in Cohen, [Cohen 78], and has later been applied to two-dimensional networks, [Weiser, Davis 80], [Johnsson et. al. 81], [Weiser, Davis 81], and extended to include explicit control [Cohen, Tyree 79], [Johnsson, Cohen 81a], and [Johnsson, Cohen 81b].

The correspondence between expressions in the mathematical notation and computational networks makes possible the use of formal symbolic transformations in the synthesis and analysis of concurrent algorithms. It is possible to proceed from a description of an algorithm in standard mathematical notation to a description of an implementation of it in space and time without leaving the domain of a mathematical notation.

The traditional mathematical notation is function-oriented, with no regard to issues related to distributing computations in space and time. By introducing the notion of storage and explicitly modeling control of computational devices a wide class of computational networks can be treated formally.

Without using a storage operator it is possible to model only combinational state-free networks composed of only memory-free functional units. However, the use of a storage operator allows for the mathematical modeling of the behavior of any computational network using both combinational logic and storage.

A major concern in design is the trade-off between time and space. A full instantiation of an algorithm <u>in space</u> is a <u>fully concurrent</u> implementation, whereas a full instantiation <u>in time</u> is a <u>fully sequential</u> implementation.

DEFINITIONS

Following [Cohen 78], the operator Z is defined by:

$$Zx(j) = x(j-1)$$

Let $Z^k = ZZ^{k-1}$

Then $Z^k x(j) = x(j-k)$

Define Z^{-1} to be the inverse of Z such that

$ZZ^{-1} = Z^{-1}Z = Z^0 = I$, where I is the identity operator.

If {x} is a sequence of data elements such that element x(j) precedes x(j+1), then j can be interpreted as time, Z as a <u>delay</u>, and its inverse as a <u>prediction</u>.

The operator Z is a model of a storage cell such as a flip-flop or a shift register cell. Z^N models a shift register of length N.

The operator Z commutes with time-independent functions:

$$ZF(U) = F(ZU) \tag{1}$$

For example, if U={u,v} then ZF(u,v) = F(Zu,Zv)

Figure 1 shows a graphical representation of this commutative distributive property.

```
                *****                      *****
                *   *<-- u                 *   *<--[Z]<-- u
  W <--[Z]<--* F *         <=>     W <---* F *
                *   *<-- v                 *   *<--[Z]<-- v
                *****                      *****
```

Figure 1: ZF(u,v) = F(Zu,Zv)

If C is a constant, then ZC=C and the following holds:

$$Z(CX) = (ZC)(ZX) = C(ZX) = CZX$$

Hence, as operators Z and C commute.

A binary multiplexor has an output signal M(X,Y,U) that is equal to one or the other of its two inputs X and Y according to a control signal U:

$$M(X,Y,U) = \begin{Bmatrix} X, \text{ if } U=1 \\ Y, \text{ if } U=0 \end{Bmatrix} = UX + U'Y \tag{2}$$

A graphical representation of the multiplexor is shown in Figure 2.

```
                    *****
                    *   *
                    *   1 <---- X
       M(X,Y,U) <----* M *
                    *   0 <---- Y
                    *   *
                    *****
                      :
                      :
                      U
```

Figure 2: Graphic representation of a multiplexer

The expression UX+U'Y is formally a sum of two products of which at least one is equal to zero. This notation is convenient in our formal derivations. By the way the control signal is introduced and defined, the selection mechanism requires no special treatment in our formalism.

When there is a need to emphasize implementation aspects we use the "#"-sign instead of the "+"-sign (e.g., UX#U'Y) to indicate the combination of two entities, of which only one is actually valid (enabled, non-zero, etc).

Throughout this note upper case symbols (like X) denote signals and operators, whereas lower case symbols (such as x) denote specific values.

THE DISCRETE FOURIER TRANSFORM

The Discrete Fourier Transform, DFT, of order N is defined by

$$y(m,k) = \sum_{j=0}^{N-1} w^{mj} \, x(k+j) \qquad m=0,1,\ldots,N-1 \tag{3}$$

where $w = w_N = e^{-2*\pi*i/N}$

One of the main ideas behind FFT algorithms, [Cooley, Tukey 65], is exploiting the following property of the w's:

$$w_N^{mj} = \begin{cases} w_N^{m(j-N/2)} & m \text{ even} \\ -w_N^{m(j-N/2)} & m \text{ odd} \end{cases} \tag{4}$$

Both decimation-in-frequency, DIF, and decimation-in-time, DIT, FFT algorithms achieve the reduction in the number of arithmetic operations by recursively computing the DFT. An N-point DFT is computed from two N/2-point DFT's.

In our derivations the computations in (3) are instantiated in time and space into N-cycles and logN units, where logN denotes $\log_2 N$ throughout this paper.

In our DIF variant j is spread in time such that values of the input signal, X, enters in normal order. As a result of this order and design desicions the output is produced in bit-reversed order.

In our DIT variant m is spread in time such that values of the output signal, Y, are produced in normal order. As a result of this order and design decisions the input has to be provided in bit-reversed order.

The bit-reversed order results from the repetitive application of chosing even-before-odd.

Decimation-in-frequency, DIF

In the following derivation of a concurrent algorithm for the DIF type FFT we use the notation $[t]\equiv t\bmod(N)$, and $\langle t\rangle = t-[t]$. Our derivation concerns the case where a sequence of DFT's are computed on contiguos windows of N samples, $N=2^n$, and the phase is chosen such that $k=\langle t\rangle=0,N,2N....$

With these definitions and assumptions equation (3) is rewritten as

$$y(m,\langle t\rangle) = \sum_{[t]=0}^{N-1} w^{m[t]}\, x(t) = \sum_{[t]=0}^{N-1} w^{m[t]}\, x(\langle t\rangle+[t]) \qquad m=0,1,...,N-1 \quad (5)$$

The summation ($\sum$) in (5) is performed over time. There are N different sums (one for each m) the order of which is not specified by (5). No component of the DFT can be available until $[t]=N-1$.

Using (4) in (3) yields:

$$y(2m,\langle t\rangle) = \sum_{[t]=N/2}^{N-1} w^{2m[t]}\, (x(\langle t\rangle+[t]) + x(\langle t\rangle+[t-N/2]))$$

or

$$y(2m,\langle t\rangle) = \sum_{[t]=N/2}^{N-1} w^{2m[t]}\, (I + Z^{N/2})\, x(\langle t\rangle+[t]) \qquad m=0,1,...,N/2-1 \quad (6)$$

and

$$y(2m+1,\langle t\rangle) = \sum_{[t]=N/2}^{N-1} w^{2m[t]}\, w^{[t]}\, (x(\langle t\rangle+[t]) - x(\langle t\rangle+[t-N/2]))$$

or

$$y(2m+1,\langle t\rangle) = \sum_{[t]=N/2}^{N-1} w^{2m[t]} w^{[t]} (I - Z^{N/2}) x(\langle t\rangle+[t]) \quad m=0,1,\ldots,N/2-1 \qquad (7)$$

Expressions (6) and (7) still defines N summations, but they are partially ordered into two sets, the set of even and the set of odd components of the DFT. Each summation ($\sum$) is to be performed over time in the interval $N/2 \leq [t] < N$. The even components are computed from one intermediate signal, the odd components from another. Indeed, as is well known, [Cooley, Tukey 65], the even and odd components are simply DFT's of length N/2 on these intermediate signals.

In the following a subscript on a signal name, such as X_N, indicate that this signal is treated as a sequence of "windows" each of length N.

We chose to compute the even components of the DFT before the odd components. Hence, one network computing DFT's of size N/2 on the intermediate signals should be sufficient in order to obtain a DFT of size N in N cycles. No order between the components within a set is yet specified. With the assumptions made so far even components can be available from time [t]=N-1 and thereafter, and the odd components from time [t-N/2]=N-1.

The two intermediate signals of (6) and (7) are computed concurrently. A new intermediate signal, $X_{N/2}$, is defined. It consists of the intermediate signal for the even components followed by the intermediate signal for the odd components of the DFT delayed by N/2 cycles.

$$x_{N/2}(t) = (I + Z^{N/2}) \, x_N(t) \qquad N/2 \leq [t] < N$$

and (8)

$$x_{N/2}(t) = Z^{N/2} \{w^{[t]} (I - Z^{N/2})\} \, x_N(t) \qquad N/2 \leq [t-N/2] < N$$

The selection mechanism implied by 8 can be implemented by a binary multiplexor controlled by a signal U_N defined as

$$u_N(t) = \begin{cases} 1 & N/2 \leq [t-N/2] < N \\ 0 & N/2 \leq [t] < N \end{cases} \qquad (9)$$

The intermediate signal $X_{N/2}$ can now be written as

$$x_{N/2}(t) = (u'_N(t)[I + Z^{N/2}] + u_N(t) Z^{N/2} [w^{[t]}(I - Z^{N/2})])x_N(t) \qquad (10)$$

```
      *****  o<---------------------- + <----o<--------------o<--- X
 X    *   *  |                          \   /                |      N
  N/2 *   0<-o                           \ /                 |
<-----* M *                               /                  |
      *   1<-o   +------+   +-----+      / \       +------+  |
      *   *  |   | N/2  |   | [t] |     /   \      | N/2  |  |
      **:**  o<--| Z    |<--| w   |<-- + <----o<---| Z    |<---o
        :        +------+   +-----+    -           +------+
        :
        U
         N
```

Figure 3: Computing $X_{N/2}$ from X_N

A network implementing (10) is shown in Figure 3.

By iterating the process that lead to (10) logN times all the components of y(*,<t>) are obtained in bit-reversed order. If DIF(N) denotes the operator that transforms the signal X_N into the signal $X_{N/2}$, then a complete DFT is performed by DIF(2)DIF(4)...DIF(N/2)DIF(N)X_N.

Each DIF(N) needs a control signal, U_N. The control signals for the different units are in phase with each other in the sense that when the signal with the lowest frequency has a 0-1 transition, so have all the others. Since it is easier to half the frequency of square waves than to double it, U_N is derived from $U_{N/2}$ unlike $X_{N/2}$ which is derived from X_N. Therefore, the control signal propagates in the opposite direction to the propagation of the data, X.

A simplified drawing of Figure 3 is shown in Figure 4, and an array for N=8 is shown in Figure 5.

```
           ***************
           *             *
X    <--*     DIF(N)     *<-- X
 N/2       *             *     N
           ***************
                  :
U ............:
 N
```

Figure 4: Computing $X_{N/2}$ from X_N

```
         ***********       ***********       ***********
         *         *       *         *       *         *
Y = X <--*  DIF(2) *<--X --*  DIF(4) *<--X --*  DIF(8) *<---- X = X
     1   *         *    2  *         *    4  *         *         8
         ***********       ***********       ***********
              :                 :                 :
              :    +---+        :    +---+        :    +---+
U ...........>:...>|f/2|...U ..>:...>|f/2|...U ..>:..>|f/2|....> U
 2                 +---+    4        +---+    8        +---+        16
```

Figure 5: A decimation-in-frequency array for N=8

Decimation-in-time, DIT

In our derivation of space-time algorithms of the DIT type FFT we order the computations such that the components of the DFT are computed in order of increasing index.

$$y_N([t],\langle t\rangle) = \sum_{j=0}^{N-1} w^{[t]j}\, x_N(\langle t\rangle+j) \qquad (11)$$

For DIT algorithms the summation in equation (11) is split into two, one containing even components of X, one containing the odd components.

$0 < [t] < N/2$

$$y_N([t],\langle t\rangle) = \sum_{j=0}^{N/2-1} w^{2[t]j}\, x_N(\langle t\rangle+2j) + w^{[t]} \sum_{j=0}^{N/2-1} w^{2[t]j}\, x_N(\langle t\rangle+2j+1) \qquad (12)$$

$N/2 < [t] < N$

$$y_N([t],\langle t\rangle) = \sum_{j=0}^{N/2-1} w^{2[t-N/2]j}\, x_N(\langle t\rangle+2j) -$$

$$- w^{[t-N/2]} \sum_{j=0}^{N/2-1} w^{2[t-N/2]j}\, x_N(\langle t\rangle+2j+1) \qquad (13)$$

By properly observing the properties of the coefficients, w, the time arguments in the right hand side of (13) have been made the same as those in (12). The two expressions define the DIT butterfly.

Each summation ($\sum$) in the above two expressions represent N/2 sums. We define a new signal, $Y_{N/2}$, that during the interval $0 \leq [t] < N/2$ carries the sums based on the even components of X_N, and during the next half window carries the sums based on the odd components of X_N.

$0 < [t] < N/2$

$$y_{N/2}([t],\langle t\rangle) = \sum_{j=0}^{N/2-1} w^{2[t]j}\, x_N(\langle t\rangle+2j)$$

$N/2 < [t] < N$

$$y_{N/2}([t],\langle t\rangle) = \sum_{j=0}^{N/2-1} w^{2[t]j}\, x_N(\langle t\rangle+2j+1)$$

Hence, the two expressions (12) and (13) can be written as

$\underline{0 < [t-N/2] < N/2}$

$$Z^{N/2}\, y_N([t],\langle t\rangle) = Z^{N/2}\, y_{N/2}([t],\langle t\rangle) + w^{[t]}\, y_{N/2}([t],\langle t\rangle)$$

or

$$Z^{N/2}\, y_N([t],\langle t\rangle) = (Z^{N/2} + w^{[t]} I)\, y_{N/2}([t],\langle t\rangle) \qquad (14)$$

$\underline{N/2 < [t] < N}$

$$y_N([t],\langle t\rangle) = Z^{N/2}\, y_{N/2}([t],\langle t\rangle) - w^{[t]}\, y_{N/2}([t],\langle t\rangle)$$

or

$$y_N([t],\langle t\rangle) = (Z^{N/2} - w^{[t]} I)\, y_{N/2}([t],\langle t\rangle) \qquad (15)$$

For each instance of time two components of the DFT are computed according to (14) and (15). Even though it might be of limited interest to sequentialize the output, we will do so in order to arrive at a ("general") unit that has one input and one output.

A sequential output is obtained by delaying the high order half of the DFT components by N/2 steps before they are output. The selection between the streams of low and high order components of the DFT is made by a multiplexor controlled by a signal U_N, defined as in equation (9).

With these assumptions the two expressions for the output signal Y_N can be unified as:

$$Z^{N/2}\, y_N([t],\langle t\rangle) =$$
$$= (u'_N(t)[Z^{N/2} + w^{[t]} I] \# u_N(t)Z^{N/2}\, [Z^{N/2} - w^{[t]} I])y_{N/2}([t],\langle t\rangle) \qquad (16)$$

A network implementing (16) is shown in Figure 6.

The network in Figure 6 contains the same components as the network in Figure 3. The complexity and control of the output unit for the DIT algorithm is the same as the complexity and control of the input unit for the DIF algorithm. The difference between the units lies in the way the elements are interconnected.

If sums based on odd components are ordered before sums based on even components the delays in the upper path of Figure 6 would instead occur in the lower path after the multiplier. In that case the DIT butterfly

```
                                           +-------+
                                           | N/2   |
       *****  o<-------------- + <---o<---| Z     |---o<-- Y
  Y    *   *  |                \   /      +-------+   |      N/2
   N   *   0<-o                 \ /                   |
 <-----* M *                     /                    |
       *   1<-o  +-------+      / \       +-------+   |
       *   *  |  | N/2   |     /   \      | [t]   |   |
       **:**  o<--| Z     |<--- + <---o<---| w     |<--o
         :       +-------+      -         +-------+
         :
         U
          N
```

Figure 6: Computing Y_N from $Y_{N/2}$

unit would essentially be a mirrored version of the DIF butterfly unit. However, the input would have to be supplied in inverted bit-reversed order.

The inverse Fourier Transform is obtained by exchanging w with its inverse and normalizing. It is readily checked that if the units in Figure 6 and Figure 3 are connected in series and one unit use w^{-1} instead of w, then the input signal (multiplied by a factor of 2) is also the ouput signal, irrespective of which unit is used as input unit.

The procedure that resulted in (16) can be applied logN times. However, it should be noticed that for instance instead of an expression for $Y_{N/2}$ an expression for $Z^{N/4}Y_{N/2}$ is obtained. By applying $Z^{N/4}$ to both sides of (16) the expression for $Z^{N/4}Y_{N/2}$ can be used in (16). Hence, eventually an expression for $Z^{N-1}Y_N$ is obtained, i.e., our notation and derivations properly accounts for the fact that no frequency component can be computed before all required data is available. The first component of Y is available at [t]=N-1.

If DIT(N) is the operator that transforms $Y_{N/2}$ into Y_N, then a complete DFT is generated by DIT(N)DIT(N/2)...DIT(2)X.

A simplified drawing of Figure 6 is shown in Figure 7, depicting the operator DIT(N).

```
              ***************
              *             *
  Y    <--*    DIT(N)     *<-- Y
   N        *             *     N/2
              ***************
                     :
                     :.......... U
                                  N
```

Figure 7: Computing Y_N from $Y_{N/2}$

An array for N=8 is shown in Figure 8.

```
         ***********       ***********       ***********
         *         *       *         *       *         *
Y = Y <--*  DIT(8) *<--Y --*  DIT(4) *<--Y --*  DIT(2) *<---- Y = X
     8   *         *    4  *         *    2  *         *       1
         ***********       ***********       ***********
              :                 :                 :
     +---+    :        +---+    :        +---+    :
U  ..|f/2|<...:<..U ...|f/2|<...:<..U ...|f/2|<...:<......... U
 16  +---+        8    +---+        4    +---+                 2
```

Figure 8: A decimation-in-time array for N=8

SUMMARY

In this paper concurrent FFT algorithms have been derived in a formal manner from the definition of the Discrete Fourier Transform and either an assumed order of the input data, or an assumed order of the output. With assumed input order the order of the output is a consequence of decisions of how to sequentialize the results of a butterfly operation. With assumed output order the order of the input data is affected by the same decisions.

Our implementations of FFT algorithms requires explicit control. The control is included in the expressions describing the computations of each unit in the array. The ratio of the frequencies of the control signal for neighboring units is two. Hence, the control signals can be obtained from the different stages of a binary counter. The ratio of frequencies of the control signals for the input unit to that for the output unit is 2/N for the DIF algorithm and N/2 for the DIT algorithm.

Different implementations of FFT algorithms can be obtained by formal transformations of expressions in our notation. The total storage requirement, the communication topology, the sequence of coefficients needed, and the control may vary.

The use of our notation suggests the use of shift registers in synchronous systems. It is strictly an ordering operator that also can be implemented in a self-timed system.

Expressions in the notation can be given an interpretation in the implementation domain, including sequencing and control.

ACKNOWLEDGEMENT

This work was sponsored by the Defense Advanced Project Agency (DARPA) under Contract No. MDA903-81-C-0335 with USC/Information Sciences Institute and Contract N00014-79-C-0597 with the California Institute of Technology. Views and conclusions contained in this paper are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

REFERENCES

[Cohen, Tyree 79]
Cohen D., Tyree V. C.
VLSI System for Synthetic Aperture Radar (SAR) Processing.
In Proceedings of SPIE, vol. 186, pages 166-177. Society for Photo-Optical Instrumentation Engineers, August, 1979.

[Cohen 78]
Cohen D.
Mathematical Approach to Iterative Computational Networks.
In Proceedings of the Fourth Symposium on Computer Arithmetic, pages 226-238. IEEE Computer Society, October, 1978.

[Cooley, Tukey 65]
Cooley J. W., Tukey J. W.
An Algorithm for the Machine Calculation of Complex Fourier Series.
Mathematics of Computation 19:297-301, 1965.

[Johnsson et. al. 81]
Johnsson, S. Lennart, Weiser, U., Cohen, D., and Davis, A.
Towards a Formal Treatment of VLSI Arrays.
In Proceedings of the Second Caltech Conference on VLSI, pages . Caltech Computer Science Department, January, 1981.

[Johnsson, Cohen 81a]
Johnsson S. L., Cohen D.
A Mathematical Approach to Modelling the Flow of Data and Control in Computational Networks.
In H. T. Kung, B. Sproull, G. Steele, editor, VLSI Systems and Computations, pages 213-225. Computer Sciences Press, October, 1981.

[Johnsson, Cohen 81b]
Johnsson S. L., Cohen D.
A VLSI Approach to Real-Time Computation Problems.
In Proceedings of SPIE, vol. 298. Society for Photo-Optical Instrumentation Engineers, August, 1981.

[Weiser, Davis 80]
Weiser, Uri and Davis, Alan L.
Mathematical Representation for VLSI Arrays.
Technical Report UUCS-80-111, University of Utah, Computer Science Department, September, 1980.

[Weiser, Davis 81]
Weiser Uri, Davis Al.
A Wavefront Notation Tool for VLSI Array Design.
In H. T. Kung, B. Sproull, G. Steele, editor, VLSI Systems and Computations, pages 226-234. Computer Sciences Press, October, 1981.

USC Workshop on VLSI and Modern Signal Processing, Los Angeles, CA, 1 - 3 Nov. 1982

Problem-oriented specification of concurrent algorithms

Armin B. Cremers and Thomas N. Hibbard

Preliminary version

September 1982

Armin B. Cremers
Computer Science Department
University of Dortmund
P.O. Box 50 05 00
D-4600 Dortmund 50
Fed. Rep. Germany

Thomas N. Hibbard
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91103
U.S.A.

Work supported in part by ONR Contract N 00014-77-C-0536 through the University of Southern California.

Introduction

VLSI challenges the programmer to take advantage of the concurrency hidden in computational problems. The ease with which a large number of processing and memory elements can be combined in a circuit makes it desirable to look for algorithmic solutions that process large amounts of information concurrently. An increase of efficiency can be expected if the algorithm arranges for a balanced distribution of work load while observing the requirement of locality, i.e. short communication paths. These properties of load distribution and information flow serve as guidelines to the designer of algorithms for VLSI.

In the past several years, a great number of algorithms in rather disjoint application areas have been discovered adequate to the properties of VLSI. Many of these algorithms have sequential counterparts that had been known for decades but had been discarded for lack of efficiency. No widely applicable methodology has evolved yet to support a review of the art of computer programming for suitable VLSI solutions to a given computational problem.

In view of the high cost of reliable VLSI implementation, and in view of the rising complexity of application demands, it is important to exercise great care at the point of algorithmic specification. Without disputing the usefulness of diagrams, commented examples, and hints at known sequential versions, the definition of a VLSI algorithm doubtlessly requires a precise notation capable of reflecting the concurrent nature of the algorithm. The algorithmic definition serves as input to the VLSI implementation cycle that starts with chip design and ends with the tested chip. In addition, the definition serves as a basis for the functional description of the device to the environment.

An important advantage of a precise and adequate notation for concurrent algorithms is that it may be executable and thus offer a potential for high-level testing and diagnosis.

The main purpose of the present paper is to suggest an executable algorithmic notation for VLSI solutions which, in many cases, may be more natural than traditional programming notation. The notation is based on the concept of a data space, introduced by the authors several years ago [4, 5] as a formal model of abstract machines. The data space notion is very close in nature to the modules of Parnas [17] and the processes of Horning and Randell [12]. A data space encapsulates control-oriented and data-oriented modelling, and thus may be a vehicle for combining the advantages of both approaches. Basically, a data space is a transition system whose states are given an explicit information structure. Each transition is computed as a function of the whole state. The model therefore includes the possibility of powerful changes to the state in a single step. This latter point has also been one of the central ideas of the applicative state transition systems suggested by Backus [2].

Initially introduced as an operational semantic model of programming, data spaces have been investigated since 1979 from the point of view of software specification. A large class of finitely specifiable data spaces has been identified and given a computable syntax. These data spaces are called syntactic or context free. Two versions of the syntax have been implemented in PL/I and PASCAL, resp., and have been successfully used as vehicles of specification in different application areas [9]. Our notation is applicative in the sense that, during the computation of a state transition there are no side effects on the state of the data space.

For the purpose of the present paper, the syntax has been extended by some new concepts. Primarily, these are subspaces, intended as an encapsulation of computational substructures, and synchronized types, a very basic communication mechanism, somewhat similar to the one of Hoare's CSP [11].

We emphasize that we are not proposing a new programming language: our notation is not designed to directly produce efficient code. On the other hand, our notation is not intended to directly generate input for an automatic verification system. In order to keep the notation simple, yet problem oriented, we had to "separate concerns." However, it is quite conceivable that our proposal could be extended in either direction.

In section 2 we convert two typical systolic array algorithms, convolution and matrix multiplication, to data spaces. In both we let synchronization be global. In the convolution example we include all the details of pipelining output and resetting for a new problem. In the second example we omit these details since they are of the same general nature as in the first example and not as intricate.

In section 3 we turn to local synchronization and introduce some additional constructs for the purpose. We show matrix multiplication again, and convert a bubble sorting array to a data space.

In section 4 we discuss, we think for the first time, that a concurrent binary search tree can be maintained in an optimal configuration (for uniform search frequencies among the keys) under random insertion for only 2 log n cycles per insertion. We first define this functionally where concurrency consists in simultaneous evaluation of the arguments of function calls, then convert that to a concurrent tree of locally synchronized processing elements.

REFERENCES

1. Armstrong, P.N., An Investigation of Sorting and Self-Sorting Memory, Final Tech. Rept. on Smart Memory Structures, Caltech 1977.

2. Backus, J., Can Programming Be Liberated from the Von Neumann Style? A Functional Style and Its Algebra of Programs, Comm. ACM 21, 613-641, Aug. 1978.

3. Browning, S.A., Algorithms for the Tree Machine, Chapter 8.4.2 of [16].

4. Cremers, A.B., Hibbard, T.N., Formal Modeling of Virtual Machines, IEEE Transactions on Software Engineering 4, 426-436, 1978.

5. Cremers, A.B., Hibbard, T.N., Functional Behavior in Data Spaces, Informatik-Forschungsbericht Nr. 37/77, Universität Dortmund; Acta Informatica 9, 293-307, 1978.

6. Cremers, A.B., Hibbard, T.N., Orthogonality of Information Structures, Informatik-Forschungsbericht Nr. 31/76, Universität Dortmund; Acta Informatica 9, 243-261, 1978.

7. Cremers, A.B., Hibbard, T.N., On the Formal Definition of Dependencies Between the Control and Information Structure of a Data Space, Theoretical Computer Science 5, 113-128, 1977.

8. Cremers, A.B., Hibbard, T.N., Data Spaces with Indirect Addressing, Informatik-Forschungsbericht Nr. 43/77, Universität Dortmund; Mathematical Systems Theory 11 151-173, 1978.

9. Cremers, A.B., Hibbard, T.N., Specification of Data Spaces by Means of Context-Free Grammar-Controlled Primitive Recursion, Informatik-Forschungsbericht Nr. 107/80, Universität Dortmund.

10. Haynes, L.S., Lau, R.L., Siewiorek, D.P., Mizell, D.W., A Survey of Highly Parallel Computing, IEEE Computer, 9-24, Jan. 1982.

11. Hoare, C.A.R., Communicating Sequential Processes, Comm. ACM 21, 666-677, 1978.

12. Horning, J.J., Randell, B., Process structuring, ACM Computing Surveys 5, 5-30, 1973.

13. Kung, H.T., Notes on VLSI Computation, CS Tech. Rept., Carnegie-Mellon University, Sept. 1980.

14. Kung, H.T., Why Systolic Architectures?, IEEE Computer, 37-46, Jan. 1982.

15. Kung, S.Y., Gal-Ezer, R.J., Arun, K.S., Wavefront Array Processor: Architecture, Language and Application, Proc. Conf. on Advanced Research in VLSI, M.I.T., Jan. 1982.

16. Mead, C., Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, Reading, Pa., 1980.

17. Parnas, D.L., A Technique for Software Module Specification with Examples, Comm. ACM 15, 330-336, 1972.

AD P[illegible]

An Overview of Signal Representations in Signal Processing Languages[*]

Gary E. Kopec

Fairchild Laboratory for Artificial Intelligence Research
Palo Alto, CA 94304

§1 Introduction

Computer science research in programming methodology has led to the view that the fundamental activity in the development of well-structured programs is the recognition of *abstractions*[1]. In particular, two general kinds of abstractions have been identified— abstract *operations* and abstract *data types*[1–4]. An abstract operation corresponds to the notion of a procedure or subroutine which is supported in most contemporary programming languages. An abstract data type consists of a set of objects plus a set of primitive operations for creating and manipulating those objects. Examples of common data abstractions include fixed and floating point numbers, character strings, arrays, records, priority queues and I/O streams[4].

Digital signal processing is based on a well-established body of mathematical theory which is explicitly used during program development. Perhaps the most distinctive features of this theory are the central roles of the concepts *signal* and *system*. This suggests that a well-structured signal processing program is one organized as a collection of "signal" and "system" abstractions. Similarly, a "signal processing programming language" is a language which provides explicit support for such abstractions.

This paper reviews three approaches to the representation of discrete-time signals as objects in programs. The first two representations, *arrays* and *streams*, are widely used in contemporary signal processing programming. The third representation was introduced in the recently-proposed signal representation language SRL[5]. SRL signals are abstract objects whose properties are explicitly designed to reflect those of the represented signals. Arrays, streams and SRL signal objects are discussed in the context of a set of signal representation criteria which are motivated by elementary observations about the mathematics of discrete-time signals. The emphasis in this paper is on the semantics of signal representation rather than on issues of time- or space-efficiency.

[*]Summary of a presentation given at the USC Workshop on VLSI and Modern Signal Processing, Los Angeles, CA, Nov. 1–3, 1982.

§2 Desiderata for Signal Representation

A set of requirements for a signal representation is motivated by three elementary observations about the mathematics of discrete-time signals and the notations commonly used to describe them. The first observation is that signals are constant values whose properties are not subject to change. This suggests that, in a program, signals should be represented by objects which are *immutable*[6]. The second observation concerns the kinds of properties that distinguish one signal from another. Mathematically, two signals are equal if they have the same domain and their values are equal at each point of their domain. This suggests that the *inquiry operations*[7] for a signal should consist of a function which identifies the domain of the signal and a function which returns the value of an arbitrary sample.

The third observation about discrete-time signals concerns the way in which signals are specified in ordinary mathematical notation. A class of signals is often defined by giving an expression for the value of the prototypical sample of the prototypical signal of the class. This expression contains a number of "free" parameters which define the space of represented signals. Each signal in the class corresponds to a specific set of values for the parameters. For example, the prototypical sine wave might be defined as the signal whose value at n is

$$\sin(\omega \cdot n + \varphi)$$

where ω and φ are parameters representing the frequency and initial phase of the wave. A particular sine wave is characterized by its specific values for ω and φ.

The mathematical notation for signals suggests that the signal objects in a program should be grouped into *signal classes*, where each class is associated with a parameterized procedure for computing the value of an arbitrary sample of any signal in the class. A particular signal of the class should be created by binding these parameters to a set of specific values.

§3 Arrays and Streams

Fig. 1 shows a set of operations for a simple multi-dimensional array abstraction. The operation **make-array** creates and returns a new array object with the specified dimensions. The elements of an array with dimensions $N_1, \ldots, N_m$ have indices $i_1, \ldots, i_m$, where $i_k \in [0, N_k)$. The operation **array-dimensions** returns a list containing the dimensions of its array argument. The operation **array-fetch** returns the value of an array element and **array-store** changes the value of an element.

The inquiry operations **array-fetch** and **array-dimensions** satisfy the requirements for signal inquiry operations presented above. However, the operation **array-store** can be used to alter the observable properties of an array and thus violates the immutability criterion. Finally, the array abstraction does not support the notion of parameterized classes of signals.

Fig. 2 shows a set of operations for a simple **stream**, or first-in-first-out (FIFO) queue, abstraction. The operation **make-stream** creates new stream objects. The operation **stream-put** appends an element to the end of a stream and **stream-get** removes an element from the front of the stream. The operation **stream-put-eos** appends a special *end of stream* token to the end of the stream. The operations **stream-is-empty** is a predicate which returns **true** if the stream currently contains no elements.

The stream inquiry operations (**stream-get** and **stream-is-empty**) do not satisfy the requirements for signal inquiry operations. For example, the "length" of a stream is not an explicitly-supported notion. Furthermore, streams are mutable and are inapproprite for multi-dimensional signals. Finally, there is no support for the concept of signal classes.

§4 SRL

The fundamental activity in SRL programming is the implementation of *signal types*. A signal type is a representation for a class of signals which share a common procedure for computing the value of a sample. Instances of a signal type are created by fixing the values of the *free variables*[7] of this procedure.

Fig. 3 lists the basic inquiry operations on SRL signal objects. The operation **signal-dimensions** returns a list containing the dimensions of its argument. The operation **signal-fetch** returns that sample of the (multidimensional) signal whose indices are **i-1**, ..., **i-n**. The **with-signal-values** form is an inquiry operation for obtaining an array containing the samples of a signal. The body of the form is a collection of expressions requiring access to the samples of signals **sig-1**, ..., **sig-n**. SRL executes the body after binding local variables **var-1**, ..., **var-n** to arrays containing the requested samples.

Fig. 4 summarizes the syntax of **defsigtype**, the SRL signal type definition form. The first argument to **defsigtype** is the name of the type being defined. The remaining arguments, consisting of alternating keywords and values, specify various optional properties shared by signals of the type. A number of these options are described below, in the context of a simple example. Fig. 5 shows an implementation of the class **sum-signal**, a representation for sums of pairs of signals.

The **:parameters** option of **defsigtype** defines the names of the parameters by which individual signals of the type are distinguished. The parameters of a **sum-signal** are **x1** and **x2**, the two signals whose sum is represented. The **:finder** option specifies the name to be given to an automatically-generated function (the signal *finder*) which returns instances of the signal type. The arguments to a finder are the values of the signal parameters. For example, if x_1 and x_2 are two SRL signal objects, a representation for their sum is returned by the finder invocation

$$(\textbf{signal-sum}\ x_1\ x_2).$$

The **:fetch** option specifies the argument list and name of a parameterized procedure (the *fetch method*) which returns a sample of any signal in the class being defined. The fetch method is invoked by the general inquiry operation **signal-fetch**. The parameters of the signal are accessed as free variables within the body of the fetch method. The fetch method of **sum-signal** computes a requested sample by fetching the corresponding samples of **x1** and **x2** and adding them.

The **:values** option (not illustrated in fig. 5) is used in array-oriented computation. It specifies the body of a 0-argument procedure, the *values method*, which generates the signal value array returned by the inquiry operation **with-signal-values**.

SRL appears to satisfy the identified criteria for signal representation. The fundamental activity in SRL programming is the implementation of signal types. The basic observable properties of a signal are the dimensions of its domain and the values of its samples. Finally, signal objects are immutable so that these properties remain fixed after a signal is created.

References

[1] B. Liskov, "An introduction to CLU", CSG Memo 136, MIT Laboratory for Computer Science, Cambridge, MA (Feb. 1976).

[2] J. Guttag, "The specification and application to programming of abstract data types", TR CSRG-59, Computer Systems Research Group, Univ. of Toronto (Sep. 1975).

[3] W. Wulf, R. London and M. Shaw, "Abstraction and verification in Alphard: introduction to language and methodology", Carnegie-Mellon Univ. Technical Report (June 1976).

[4] B. Liskov, R. Atkinson, T. Bloom, E. Moss, C. Schaffert, R. Scheifler and A. Snyder, "CLU reference manual", TR-225, MIT Laboratory for Computer Science, Cambridge, MA (Oct. 1979).

[5] G. Kopec "The signal representation language SRL", submitted for publication to the *IEEE Trans. on Acoustics, Speech and Signal Processing.*

[6] B. Liskov, A. Snyder, R. Atkinson and C. Schaffert, "Abstraction mechanisms in CLU", *CACM* **20**(8) (Aug. 1977).

[7] B. Liskov and V. Berzins, "An appraisal of program specifications", in *Research Directions in Software Technology*, MIT Press, Cambridge, MA (1979).

```
(make-array n-1 ... n-m)
(array-dimensions x)
(array-fetch x i-1 ... i-m)
(array-store val x i-1 ... i-m)
```

Fig. 1 Operations of simple array abstraction

```
(make-stream)
(stream-put val x)
(stream-get x)
(stream-put-eos x)
(stream-is-empty x)
```

Fig. 2 Operations of a simple stream abstraction

```
(signal-dimensions signal)
(signal-fetch signal i-1 ... i-n)
(with-signal-values ((var-1 sig-1)
                     .
                     .
                     .
                     (var-n sig-n))
  body)
```

Fig. 3. SRL signal inquiry operations

```
(defsigtype name
    :parameters (par-1 ... par-n)
    :finder name
    :fetch ((i-1 ... i-n) body)
    :a-kind-of parent-type
    :init init-form
    :values values-form
    :variables (var-1 ... var-n))
```

Fig. 4. Summary of SRL signal type definitions.

```
(defsigtype sum-signal
  :a-kind-of basic-signal
  :parameters (x1 x2)
  :finder signal-sum
  :init (setq-my dimensions (min
                              (signal-dimensions x1)
                              (signal-dimensions x2)))
  :fetch ((n)
          (+ (signal-fetch x1 n)
             (signal-fetch x2 n))))
```

Fig. 5. Definition of sum-signal, a simple binary signal combination type.

Multi-Computer Parallel Arrays, Pipeline Arrays, and Pyramids

Leonard Uhr
Department of Computer Sciences
University of Wisconsin

This paper describes and examines parallel arrays, pipeline arrays and pyramids of arrays - their architecture, the kinds of image processing and pattern perception algorithms and programs for which they appear to be appropriate, and their appropriateness for VLSI implementations.

Parallel Arrays of Very Large Numbers of Processors

A number of 2-dimensional arrays have been built in recent years, or are now being completed. These include the 64 by 64 DAP ("Distributed Array Processor"), the 96 by 96 CLIP4 ("Cellular Logic Information Processor"), and the imminent 128 by 128 MPP ("Massively Parallel Processor").

Each of such an array´s thousands of computers executes the same instruction, but on a different set of data. The data to be processed are input to a large array, ideally the same size as the array of computers, so that each computer has one sub-set of those data in its own memory, for example, one pixel from the total image. Then each computer operates on data stored in its own memory, directly linked near-neighbor computers´ memories.

For example, a basic CLIP4 machine-language instruction can have every computer fetch (in parallel) and operate upon information from its own memory and also from any or all of the 8 immediately surrounding computers´ memories - that is, from the 3 by 3 "window" surrounding it.

Today´s arrays have from 4,000 to 16,000 computers. But well within current technology and current economics is the capability to build much larger arrays, with on the order of 256 by 256, 512 by 512 or even 1,024 by 1,024 computers.

Assembly-line Pipe-line Arrays of Computers (or Processors)

Several "pipe-lines" of processors or computers have been built, much in the spirit of an assembly-line of workers. Each processor in the pipe continues to execute the same instruction on a sequence of data flowing through that pipe. This means that if the same sequence of instructions is to be executed on a large number of different pieces of data (the case whenever the data stored in an array are to be operated on, as in image processing, pattern perception and 3-dimensional modelling) a pipe-line as long as the sequence of instructions can be built, and data (e.g., the cells of an array) flowed through the pipe-line´s processors.

If the pipe-line has N processors, then the program will execute up to approximately dN times as fast as a 1-processor computer. (d is the often appreciable saving from not having to fetch and decode the next instruction, since each processor fetches only once, and keeps executing the same instruction). The longest pipeline built to date is the Cytocomputer, with 118 processors specialized for image processing.

Pyramids of Arrays

An especially attractive structure is a pyramid whose base is a large array, with successively smaller arrays on top of it.

An image array is input to the memory stores at the base of the pyramid. Each processor links to its near-neighbors in its own array (usually 4 square neighbors, or 8 square-and-diagonal neighbors, or 6 hexagonal neighbors), and to nearby offspring in the next-larger array below it (usually a 2 by 2 sub-array of 4 offspring) and to one or several parent nodes in the next-smaller array above. A pyramid can also be conceived of and constructed as a tree linked at its buds to an array and (possibly) with its interior nodes linked to nearest-neighbor siblings to form interior arrays at every level (often called "ply").

Pyramids transform, converge and merge together information as the image is pipelined from the base image array to the apex. They also effect an important reduction in the distances over which information is transferred (often called "message-passing" in today's discussions of computer networks). For example, a 1,024 by 1,024 4-square-connected array needs 2,047 operations to send data or any other kind of message from one of its corners to the opposite corner. But when this array is made into the base array of a pyramid only 20 operations are needed, since the message can be passed up to the apex of the pyramid and then back down. To state this more generally, whereas the diameter (that is, the worst-case message-passing distance) of an NxN array is O(N) the diameter of a pyramid that includes that array is O(logN).

The Appropriateness of Arrays and Pyramids for VLSI

The processors used in arrays and in pyramids of arrays are routinely kept extremely simple. In almost all cases 1-bit processors with from 100 to 800 gates have been used. In order to achieve 4 or more orders of magnitude increases in speed and power by using increasingly large numbers of processors in parallel, their architects have opted to use the simplest possible 1-bit processor, executing K-bit-serial operations to process K-bit numbers or strings.

The amount of memory each such processor needs appears to be a function of the total amount of memory needed to handle the image or other large sets of data given the system. Therefore each processor appears to need relatively small amounts of memory (present implementations have a few thousand bits of memory per processor). Today 4, 8 or even more processors are fabricated on a single VLSI chip. Because of the highly iterated micro-modular design of such systems it should soon be possible to fabricate hundreds, or even thousands, of processors, each with its own memory, on a single chip.

Parallel Implementation of Likelihood-Based Estimation and Detection

Donald W. Tufts and Louis L. Scharf

Department of Electrical Engineering
University of Rhode Island
Kingston, Rhode Island 02881

ABSTRACT

Old and new information is presented about the implementation of the detection of signals, the estimation of signals, and the estimation of parameters of signals. In all cases the processing consists of exact or approximate calculation of the logarithm of a likelihood function. The form of the likelihood is based on Gaussian statistics, even when the signal and interference are not Gaussian random functions.

For complex-valued Gaussian data the calculation of log-likelihood is the evaluation of a quadratic form. Interesting questions about the implementation of likelihood calculations are shown to be questions about the computation of $(Q^{-\frac{1}{2}})^{*} \overline{X}$, where $Q^{-\frac{1}{2}}$ is the square root of the inverse of a covariance matrix Q; the asterisk denotes a complex conjugate transpose; and $\overline{X}$ is column vector of observed data.

Two forms of the likelihood ratio detector for zero-mean, complex Gaussian vector signals in zero-mean, complex Gaussian vector noise are considered. These two forms are chosen for presentation and analysis, because, for these forms, one is naturally led to parallel-processing implementations of the detectors. It is argued that these particular detector implementations can be supplemented by certain estimators to provide good detection performance when little is known about the signal or noise covariance matrices and when the signal or noise are not Gaussian.

This work was supported by the Probability and Statistics Program of the Office of Naval Research.

SUMMARY

Suppose that the probability density of an observed random column vector $\overline{X}$ depends in a known way on a vector, $\overline{V}$, of parameters. And let us assume that we wish to estimate $\overline{V}$ based on $\overline{X}$. If little is known about the prior (before observation of $\overline{X}$) probability density of $\overline{V}$, then we may model this uncertainty by assuming that this prior probability density is constant over a wide range of $\overline{V}$. In this case a maximum-likelihood estimate of $\overline{V}$ is a vector $\overline{V}^1$ for which the logarithm of the likelihood function, $\log P(\overline{X},\overline{V})$, attains its largest value. The likelihood function, $P(\overline{X},\overline{V})$, is the probability density of the observation vector, $\overline{X}$, conditioned on a given vector, $\overline{V}$, of parameter values.

Maximum-likelihood estimates have many desirable attributes. For example, under certain general conditions, the maximum-likelihood estimate, $\overline{V}^1$, satisfies a wide class of useful criteria (1,2). For example, if $\overline{F}(\overline{V})$ is a one-to-one transformation of $\overline{V}$, then $\overline{f}(\overline{V}^{1})$ is a maximum likelihood estimate of the transformed vector (9).

When constructing likelihood for complex normal data, one is led to the study of Hermitian quadratic forms of the following type:

$$\ell = \overline{X}^{*} Q^{-1} \overline{X} \quad \text{(* denotes Hermitian transpose)} \tag{1}$$

The matrix Q is a non-negative definite correlation matrix. When it is singular, we replace Q^{-1} with the pseudo-inverse $Q^{\#}$:

$$Q^{\#} Q Q^{\#} = Q^{\#} \tag{2}$$

It is always possible to write ℓ as follows:

$$\ell = \overline{u}^{2} \quad ; \quad \overline{u}^{2} = \overline{u}^{*} \overline{u} \tag{3}$$

$$u = Q^{-*/2}\overline{X} \; ; \quad Q^{-*/2} = (Q^{-\frac{1}{2}})^{*} \quad ; \quad Q^{-1} = Q^{-\frac{1}{2}} Q^{-*/2} \tag{4}$$

Typically the correlation matrix is parameterized by unknown parameters such as autoregressive-moving average (ARMA) parameters, coefficients for orthonormal expansions, etc.. A subset of these parameters contains parameters of real interest and a disjoint subset contains nuisance parameters. In any case we call the unknown parameterization $\overline{V}$ and write

$$\ell(\overline{V}) = u(\overline{V})^{2} \tag{5}$$

$$\overline{u}(\overline{V}) = Q^{-*/2}(\overline{V})\overline{X} \tag{6}$$

All are interesting questions about the implementation of likelihood are really just interesting questions about how to compute $Q^{-*/2}(\overline{v})\overline{X}$.

Different decompositions of the matrix Q^{-1} can lead to different parallel computations of $Q^{-}(\theta)^{*/2}\overline{X}$. As examples, consider the singular-value (or eigenvalue) decomposition (12)

$$Q^{-1} = U \Sigma W^{*} = (U\Sigma^{\frac{1}{2}})(\Sigma^{\frac{1}{2}}W^{*}) = Q^{-\frac{1}{2}}Q^{-*/2} = \Sigma\, \sigma(k)\, \overline{u}(k)\, \overline{w}^{*}(k) \tag{7}$$

where Σ is a diagonal matrix with kth element $\sigma(k)$ U and W are unitary matrices with column vectors $\overline{u}(k)$ and $\overline{w}(k)$ and the Gohberg-Semencul (13,14) formula for the case in which Q is a Toeplitz matrix

$$Q^{-1} = A^{*}A - B^{*} B \tag{8}$$

where A and B are lower-triangular Toeplitz matrices.

For the continuous-time case, Kailath, Levy, Ljung, and Morf (14) recommend use of formulas analogous to 8 rather than those analogous to 7. They argue that there can be a very large number of terms in the summation analogous to formula 7. They also point out that, if Q is Toeplitz, there are convenient recursive schemes for computing the A and B matrices and for updating them as more data is observed. They state that the computation and updating of the eigenvectors of (7) may be quite complicated.

We argue below that there are compensating advantages for the approach of formula 7, especially for the adaptive case in which the statistics of the observed data are not completely known.

Now let us consider the role of the likelihood function in hypothesis testing. To be specific we assume that, under each of two hypotheses, the observed vectors are zero-mean, complex, Gaussian vectors. The elements of the parameter vectors under the two hypotheses, are the elements of covariance matrices of the observed elements of data under the hypotheses.

Let us further assume that (a) the hypotheses are based on the absence or presence of a Gaussian signal in ever-present Gaussian noise and (b) the signal vector, $\bar{s}$, and the noise vector, $\bar{n}$, are uncorrelated. Then, under each hypothesis, we have the following forms for the covariance matrix of the observed vector $\bar{x}$.

$$H_0 \text{ (signal absent)} \quad : \quad \mathrm{Cov}(\bar{x}) = N \tag{9a}$$

$$H_1 \text{ (signal present)} \quad : \quad \mathrm{Cov}(\bar{x}) = S + N \tag{9b}$$

According to various reasonable criteria for decision making - such as minimum Bayes risk, minimum probability of error, and minimum probability of one type of error for a constraint on the probability of the other type of error - the processing task is to compute the difference of the logarithms of two likelihood functions. This is equivalent to computing the likelihood ratio test statistic (3,4).

For the case of the two Gaussian hypotheses of formula 9, the difference of the two log-likelihood functions takes the form

$$P = \bar{x}^* N^{-1} \bar{x} - \bar{x}^* (S + N)^{-1} \bar{x} = \bar{x}^* \left[N^{-1} - (S + N)^{-1} \right] \bar{x} \tag{10}$$

By simultaneous diagonalization of the signal and noise covariance matrices, S and N, R.N. McDonough (4), using earlier results of N.R. Goodman (5), expressed the test statistic, P, in terms of the eigenvalues and eigenvectors of the generalized signal-to-noise ratio matrix SN^{-1}. This expression is

$$P = \sum_{k=1}^{r} \left(\frac{d(k)}{1 + d(k)} \right) \left(\bar{a}_k^* \bar{x} \right)^2 \tag{11}$$

in which r is the rank of the signal covariance matrix S (assuming that the noise covariance matrix, N, has full rank) and $d(k)$, for k = 1, 2,...r, is the set of non-zero eigenvalues of SN^{-1} and $\bar{a}_k$ is the corresponding set of column eigenvectors.

We note that the r inner products of vectors, $\bar{a}_k^* \bar{x}$, k = 1, 2,...r, can be computed simultaneously in component processors of a multiprocessor system.

An important special case of formulas 10 and 11 has been further analyzed by Claus, Kadota, and Romain (6) and by Brooks and Reed (7). That is the practically interesting case in which the signal correlation matrix has rank one. In this case the required computation has the form

$$P = (N^{-1} \bar{x})^* \bar{y} \tag{12}$$

in which v is the unique signal eigenvector which represents the signal covariance matrix S by the outer product formula

$$S = \sigma_s^2 \; vv^* \tag{13}$$

in which σ_s^2 is a positive, real number.

Suppose that N is the covariance matrix of zero-mean, correlated-plus-white noise.

$$N = \sigma^2 I + Q = \sigma^2 \quad I + \sigma^{-2} Q \tag{14}$$

in which σ^2 is a positive, real number, I is an identity matrix, and Q is the correlation matrix of the correlated noise.

Claus, Kadota, and Romain (6) studied this case, in which there exists a correlated noise component. By application of matrix-inversion identities they expressed N^{-1} as follows:

$$N^{-1} = \sigma^{-2} \left[I + \sigma^{-2} Q \right]^{-1} \tag{15a}$$
$$= \sigma^{-2} \left[I + \sigma^{-2} E\, G\, E^* \right]^{-1} \tag{15b}$$
$$= \sigma^{-2} \left\{ I - E \left[I + \sigma^2 G^{-1} \right]^{-1} E^* \right\} \tag{15c}$$
$$= \sigma^{-2} \left\{ I - E\, H\, E^* \right\} \tag{15d}$$

in which $I + \sigma^2 G^{-1}$ is a diagonal matrix with jth element down the diagonal given by $(\lambda_j + \sigma^2) / \lambda j$, where λj is the jth eigenvalue of Q. And H is a diagonal matrix with jth element down the diagonal given by $\lambda j/(\lambda j + \sigma^2)$. The columns of the matrix E are the eigenvectors of the correlated noise covariance matrix Q.

Using 15d we can rewrite formula 12 in the forms

$$P = \sigma^{-2} \quad ([I - E\, H\, E^*] \,\overline{x})^* \; \overline{v} \tag{16a}$$

and

$$P = {}^{-2} \quad [\overline{x} - E\, H\, E^* \,\overline{x}]^* \; \overline{v} \tag{16b}$$

The column vector $E\, H\, E^* \,\overline{x}$ is an estimate of the correlated noise. This correlated noise estimate can be written out more explicitly as a linear combination of the eigenvectors of the covariance matrix Q of the correlated noise.

$$E\, H\, E^* \,\overline{x} \;=\; \sum_{k=1}^{m} \left(\frac{\lambda k}{\sigma^2 + \lambda k} \right) (\overline{e}_k^* \,\overline{x})\, \overline{e}_k \tag{17}$$

in which $\overline{e}_k$ is the kth eigenvector of the rank-m noise correlation matrix Q, and $\overline{e}_k^* \,\overline{x}$ is the inner product of the observed data vector with the kth eigenvector. Again, as in formula 11 above, we can directly observe the potential for parallel processing. The innerproducts can be simultaneously and separately computed.

A scale invariant version of the above Gauss-Gauss detection problem has been considered by Scharf and Lytle (10,11). Such results can be used to construct a constant false-alarm rate (CFAR) detector. The resulting test statistics are normalized versions of formulas 11 and 16.

Above we considered the reduction in dimensionality of likelihood computations which result from the practically interesting cases of a low-rank of S and a strong, low-rank component of N. Kadota and Sheep have considered this reduction in dimensionality from the different point of view of a best low-rank approximation to the computations (15).

Now let us depart from the above tutorial discussion to consider the case in which the pertinent correlation functions are incompletely known. To be specific let us consider the special case of a rank-one signal covariance and the signal-detection test statistic of 12. We also assume that the noise covariance consists of a strong rank-one interference component plus an identity matrix, which is a special case of 14. The test statistic of 16 can then apply, and there is only one term in the summation of 17. However, because $\overline{v}$ and λ, and $\overline{e}$, are not known, we cannot directly evaluate 16. But, in the interesting case of strong interference, the best, least-squares, rank-one approximation to a linear-prediction data matrix or a corresponding covariance matrix estimate provides a nearly optimum estimate of the interference waveform or of $\lambda\overline{e}$, respectively. In this manner the interference estimate required in 16 can be formed. Then different hypotheses about $\overline{v}$ can be tested using a generalized likelihood ratio test.

In 1936, Eckart and Young presented the derivation of a procedure for finding the best lower rank approximation to a given matrix (16).

The starting point for our discussion is the singular value decomposition (SVD) of a rectangular matrix A which has real or complex entries. Eckart and Young (17) showed that the SVD of such a matrix A can be used to find an approximant to A of lower rank. The SVD of the matrix A can be specified by the following product of three matrices (18):

$$\underset{m \times n}{A} = \underset{m \times m}{U} \; \underset{m \times n}{\Sigma} \; \underset{n \times n}{V^*} \qquad (18)$$

The dimensions of each matrix are written below the matrix. The matrices U and V are unitary, and Σ is rectangular diagonal matrix of the same size as A with real, nonnegative diagonal entries. These diagonal entries, called the singular values of A, are conventionally ordered in decreasing order with the largest in the upper lefthand corner. These singular values are the nonnegative square roots of the eigenvalues of A^*A and AA^*. The asterisk is used to denote the complex conjugate transpose of a matrix.

The theorem of Eckart and Young (16) can be stated as follows:

Let A be an m x n matrix of rank r which has complex elements. Let S_p be the set of all m x n matrices of rank $p < r$. Then for all matrices B in S_p

$$\left| A - \underline{A} \right| \le \left| A - B \right| \qquad (19)$$

where

$$\underline{A} = U\underline{\Sigma}V^* \qquad (20)$$

and $\underline{\Sigma}$ is obtained from the matrix Σ of (18) by setting to zero all but its p largest singular values. The matrix norm of (19) is the Frobenius norm. That is

$$\left| A - B \right|^2 = \mathrm{tr} \left[(A - B)^* \, (A-B) \right] \tag{21}$$

Hence in words, A is the best least squares approximation of lower rank p to the given matrix.

Because of the intimate connection between least squares approximation and maximum likelihood and maximum posterior probability signal estimation in Gaussian noise, the applicability of the Eckart-Young theorem should not be surprising.

The Eckart-Young theorem can be used to improve the ill-conditioned nature of signal parameter estimation via linear prediction by replacing the raw data matrix by a "cleaned-up" data matrix of lower rank (8, 19, 20, 21).

To apply the Eckart-Young theorem it is not necessary that the signal be deterministic nor that the signal matrix be of less than full rank. What is important is that the signal matrix be of approximately lower rank in the sense that, with high probability, the signal-only matrix can be well approximated by a matrix of lower rank. This will happen for a random signal when its estimated correlation matrix has a few eigenvalues which are significantly larger than the others.

REFERENCES

1. H. Cramer, Mathematical Methods of Statistics, Princeton University Press, 1946, pp. 473-506.

2. A. Viterbi, Principles of Coherent Communication, McGraw-Hill, 1966, Appendices B and C.

3. H. Cramer, Op. Cit., pp. 525-535.

4. R.N. McDonough, "A Canonical Form of the Likelihood Detector for Gaussian Random Vectors," Journal of the Acoustical Society of America, Vol. 49, pp. 402-406, 1971.

5. N.R. Goodman, "Statistical Analysis Based on a Certain Multivariate Complex Gaussian Distribution," Ann. Math. Stat., 34, pp. 152-177, 1963.

6. A.J. Claus, T.T. Kadota, and D.M. Romain, "Efficient Approximation of a Family of Noises for Application in Adaptive Spatial Processing for Signal Detection," IEEE Transactions on Information Theory, Vol. IT-26, pp. 588-595, September 1980.

7. L. W. Brooks and I.S. Reed, "Equivalence of the Likelihood Ratio Processor, the Maximum Signal-to-Noise Ratio Filter, and the Wiener Filter," IEEE Transactions on Aerospace and Electronic Systems, pp. 690-692, Sept. 1972.

8. D.W. Tufts and R. Kumaresan, "Data-Adaptive Principal Component Signal Processing," Proceedings of the 19th IEEE Conference on Decision and Control, pp. 949-954, Dec. 1980.

9. T.W. Anderson, An Introduction to Multivariate Statistical Analysis, John Wiley and Sons, Inc., New York, NY, p. 48, 1958.

10. L.L. Scharf and D.W. Lytle, "Signal Detection in Gaussian Noise of Unknown Level: An Invariance Application," IEEE Transactions on Information Theory, Vol. 17, pp. 404-411, July 1971.

11. L.L. Scharf, "Invariant Gauss-Gauss Detection," IEEE Transactions on Information Theory, Vol. IT-19, pp. 422-427, July 1973.

12. V.C. Klema and A.J. Laub, "The Singular Value Decomposition: Its Computation and Some Spplications," IEEE Trans. Automatic Control, Vol. AC-25, pp. 164-176, April, 1980.

13. I.C. Gohberg and A.A. Semencul, "On the Inversion of Finite Toeplitz Matrices and their Continuous Analogs," Math, Issled (Russian) No. 2, pp. 201-233, 1972.

14. T. Kailath, B.C. Levy, L. Ljung, and M. Morf, "Fast Time-Invariant Implementations of Gaussian Signal Detectors," IEEE Transactions on Information Theory, Vol. IT-24, pp. 469-477, July 1978.

15. T.T. Kadota and L.A. Shepp, "On the Best Finite Set of Linear Observables for Discriminating Two Gaussian Signals," IEEE Transactions on Information Theory, April 1967.

16. C. Eckart and G. Young, "The Approximation of One Matrix by Another of Lower Rank," Psychometrika, Vol. 1, pp. 211-218, 1936.

17. "A Principal Axis Transformation for Non-Hermetian Matrices," Bull. Amer. Math. Soc., Vol. 45, pp. 118-121, 1939.

18. C. L. Lawson and R.J. Hanson, Solving Least Squares Problems, Englewood Cliffs, NJ, Prentice-Hall, 1974.

19. D.W. Tufts and R. Kumaresan, "Frequency Estimation of Multiple Sinusoids: Making Linear Prediction Perform Like Maximum Likelihood," Proc. IEEE, Sept. 1982.

20. "Singular Value Decomposition and Improved Frequency Estimation Using Linear Prediction," IEEE Trans. Acoust., Speech, Signal Process., Vol. ASSP-30, Aug. 1982.

21. R. Kumaresan and D.W. Tufts, "Singular Value Decomposition and Spectral Analysis," in Proc. 1st ASSP Workshop on Spectral Estimation, Hamilton, Ont., (Aug. 17 and 18, 1981) pp. 6.4.1-6.4.12.

AD P002 [illegible]

Digital Signal Processing Structures for VLSI

Richard A. Roberts, Clifford T. Mullis
Department of Electrical Engineering
University of Colorado
Boulder, CO 80309

1. Introduction

Digital signal processing (DSP) encompasses a variety of tasks which include linear (digital) filtering, spectral estimation, signal generation, correlation, time varying or adaptive filtering and other similar tasks. This work is directed at obtaining special purpose structures fo. DSP tasks that are well suited for VLSI implementation.

In the sequel the terms realization and implementation are used in the following sense. A realization or algorithm for a DSP problem is a specification of the task in terms of a (primitive) signal flow graph (SFG). There are, in general, an infinite number of SFG's that can realize a given task. The freedom one has in choosing a particular realization can be used to optimize some performance criterion that is generally related to the complexity of the hardware. For example, if complexity is measured by the number of multiplies per output sample, we choose a realization that minimizes the number of multiples or total arithmetic operations. An implementation is the hardware specific for a given realization. (Hardware in this context refers to a VLSI design.) There are, again, an infinite number of implementations for a given realization. The choice of an implementation is generally based on optimizing data throughput and some measure of hardware complexity.

In order to illustrate our ideas for implementing DSP tasks in VLSI we shall limit our discussion to the task of digital filtering. The general principles and parameters of design we advocate apply to all DSP tasks. However, each task has its own unique characteristics which dictates the final VLSI design.

2. Parameters for VLSI

In VLSI design the primary resource is chip area. For a given DSP problem we wish to conserve chip area and maximize data throughput. We are not explicitly concerned with the portion of area given to logic or to interconnections so long as the total area dedicated to the task is minimized or, at least, used efficiently. Thus, to a first approximation, it seems reasonable to use chip area as the measure of hardware complexity for a given realization and implementation.

To optimize chip area utilization and maximize data throughput there are at least three primary parameters in the VLSI design of a DSP problem. They include:

(i) modularity and regularity of the design
(ii) concurrency of computation
(iii) finite length register effects of the DSP realization.

Modularity and Regularity

Modularity and regularity of the realization and implementation helps to reduce interconnection area and also simplifies the original design by breaking the realization into small, easily designed subunits. A regular and modular structure with local data transfers and local processing sites will be simpler to design and easier to verify and check. Moreover, if desired, redundancy can be more easily built into the final design.

Regularity in the underlying structure of a VLSI design is currently being exploited in gate arrays to reduce the design time for VLSI chips. The gate array type of design does not impose regularity and modularity on the algorithm or realization of a DSP task. Modularity and regularity should be imposed at a functional level as, for example, in Kung's work on systolic arrays. How does one define a good modular and regular realization for a DSP task into an equivalent description which also infers the geometry of the hardware? We have attempted to answer this question by reformulating the DSP task in terms of a matrix product which transforms the input sequence u into the output y. We then perform factorizations and rearrangements of the original matrix (or matrices) to structure the geometry of the hardware. By studying the structure of the individual matrices after one or more factorizations we attempt to infer the modularity and regularity in the algorithm of interest. The process of factorization is now the key step.

Regularity introduced at the gate level may be useful to reduce manufacturing costs and to simplify the design process for present day VLSI designs. However, it is not the path that leads to fundamental breakthrough in the use of VLSI for highly computational tasks.

Parallelism

The emphasis on modularity and regularity in the structure of the algorithm introduces into the structure an implicit parallelism. By requiring local data transfers, arithmetic processing must be distributed uniformly over the entire structure. The requirement of local data transfers eliminates the possibility of long data busses into and out of a central arithmetic unit.

Finite Register Effects

In digital implementations of DSP algorithms effects due to finite length registers always degrade the performance of the implementation. This degradation can always be reduced to as small a level as desired by increasing the register length containing the parameters and internal states of the algorithm. This solution requires more chip area and, in general, reduces the chip area available for logic and communication. Thus we can always trade signal quality for area.

The most effective way to minimize finite register effects is to find new realizations for a given DSP task. In some recursive algorithms certain good realizations are orders of magnitude better than others. This means that for the same signal quality a good realization can use (significantly) shorter word length registers. Thus a good realization can be implemented in less area. It is therefore less complex.

Realizations that minimize finite register effects conserve chip area by reducing register lengths. In general, these realizations do not reduce logic and may, in fact, actually increase the amount of logic needed. Thus there is a crossover point between the reduction of register length and the increase in the required logic needed to perform a given task. In digital filtering it is a function of filter bandwidth. As bandwidth decreases, finite register effects become more important.

3. An Example - A VLSI Design for Digital Filtering

Digital filtering is a task which is described externally by the linear difference equation

$$y_k = \sum_{i=0}^{m} a_i u_{k-i} - \sum_{i=1}^{n} b_i y_{k-k} \qquad (1)$$

where u_k are input samples, y_k are output samples, and a_i and b_i are parameters of the filter. Previous work has shown [1-4] that certain classes of realizations possess good finite register effects. One such class is the orthogonal structure of which the normalized all-pole lattice of Markel-Gray is an example [5-6] and is shown in Figure 1 for the second-order case (n=2).

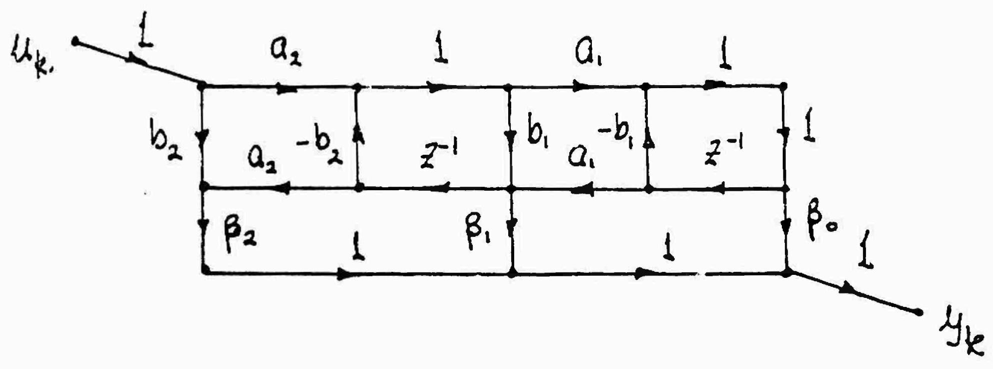


Figure 1

This lattice realization can be formulated as a matrix product in which each matrix of the product computes one of the rotations defined by the parameters a_i, b_i, $i=1,2$. By using a matrix factorization and a communitivity property on certain special forms we can obtain a new highly modular and regular structure depicted in Figure 2.

In this realization the filtering process in (1) is expressed in terms of computations of the form

$$w_i = \alpha_{1_i} x_{1_i} + \alpha_{2_i} x_{2_i}, \quad i = 1,2,... \tag{2}$$

where $\alpha_{1_i}, \alpha_{2_i}$ are constants x_{1_i}, x_{2_i} are intermediate stored variables of the filter. The filter is thus realized by a collection of modules that compute (2) and that are arranged geometrically in a highly regular structure. There are several advantages to this realization. The interconnections between modules are local connections. The modules compute such a simple form that one can implement the computation locally within each module. The word rate of the filter is determined by the module word rate. Each module works independently and so the structure is fully pipelined. The word rate of the filter is independent of the order of the filter. All modules are identical in form - only the parameters α_{1_i}, α_{2_i} $i=1,2,...$ change with the filter transfer function.

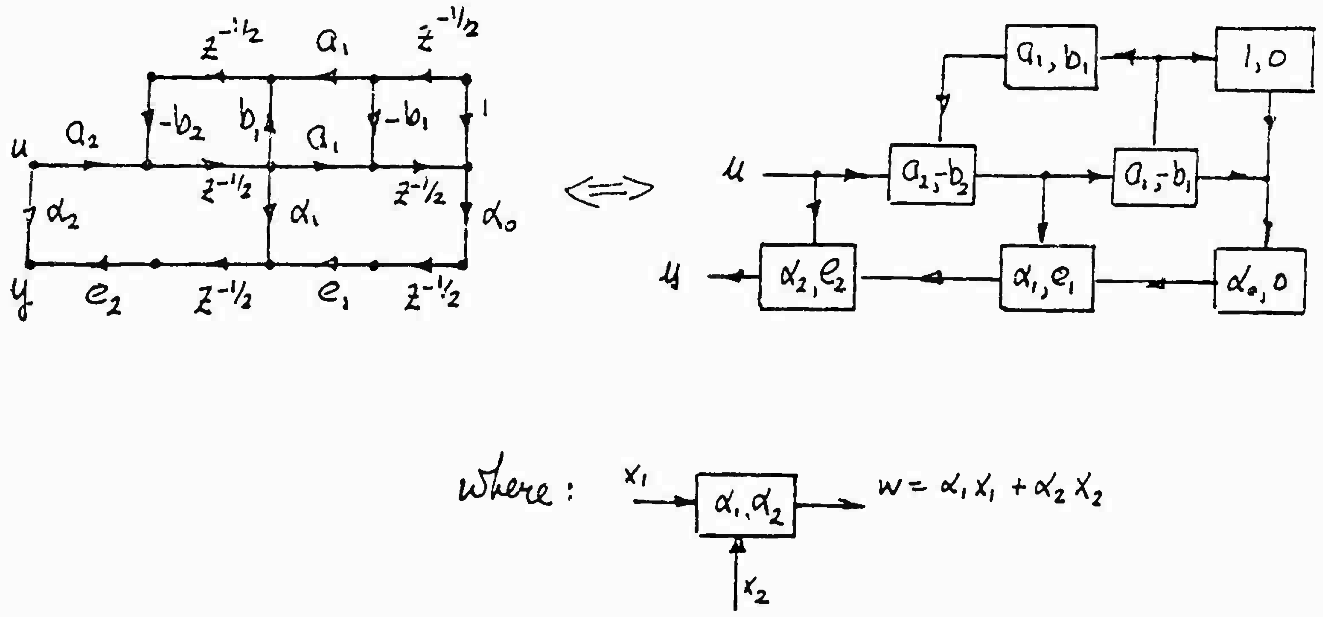


Figure 2

Generalizations of the Modular Lattice Structure

The modular lattice described here is one structure in a large class of highly modular and regular structures known as orthogonal structures. These structures all possess good finite arithmetic effects which implies they can be implemented using short register lengths. These structures are obtained by factoring a state variable representation of the digital filter into a product of matrices that represent each stage of the filter. By rearrangement of the matrices using communativity one can obtain various forms of the filter. These various forms represent trade-offs between the order of the inner produce required at each stage, the number of computational cycles needed at each stage, and the number of unit delay registers required in the structure.

Summary

The modular lattice structure proposed here for digital filtering is one realization and implementation that appears to be well suited for possible VLSI implementation. It is highly regular and modular, uses only local interconnections, it is fully pipelined, possesses excellent finite register properties, and can be implemented with many simple arithmetic processing sites. The promise of VLSI for DSP is to obtain good algorithms for particular tasks. In the same way the FFT algorithm revolutionized the computation of the DFT, there are realizations and implementations for other DSP problems that are well-suited for VLSI.

References

[1] C. T. Mullis and R. A. Roberts, "Synthesis of Minimum Roundoff Noise Fixed Point Digital Filters," IEEE Trans. on Circuits and Systems, Vol. CAS-23, No. 9,

September 1976.

[2] C. T. Mullis and R. A. Roberts, "Roundoff Noise in Digital Filters: Frequency Transformation and Invariants," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-24, No. 6, December 1976.

[3] L. B. Jackson, A. G. Lindgren, and Y. Kim, "Optimal Synthesis of Second-Order State Space Structures for Digital Filters," IEEE Trans. on Circuits and Systems, Vol. CAS-26, No. 3, March 1979.

[4] C. W. Barnes, "Roundoff Noise and Overflow in Normal Digital Filters," IEEE Trans. on Circuits and Systems, Vol. CAS-26, No. 3, March 1979.

[5] A. H. Gray, Jr. and J. D. Markel, "Digital Lattice and Ladder Filter Synthesis," IEEE Trans. on Audio and Electroacoustics, Vol. AU-21, December 1973.

[6] A. H. Gray, Jr. and J. D. Markel, "A Normalized Digital Filter Structure," IEEE Trans. on Acoutics, Speech and Signal Processing, Vol. ASSP-23, June 1975.

AD P002612

LINEAR OR SQUARE ARRAY FOR EIGENVALUE AND SINGULAR VALUE DECOMPOSITIONS?[1]

S.Y.Kung and R.J.Gal-Ezer

Department of Electrical Engineering

University of Southern California

Los Angeles, California 90089

INTRODUCTION

For many signal and image processing applications, such as high resolution spectral estimation, image data compression, etc., [1,2,3] eigenvalue and singular value decompositions have emerged as extremely powerful and efficient computational tools. As far as the symmetric eigenvalue problem[2] is concerned, "QL and QR algorithms ... have emerged as the most effective way of finding all the eigenvalues of a small symmetric matrix. A full matrix is first reduced to tridiagonal form by a sequence of reflections and then the QL [QR] algorithm swiftly reduces the off diagonal elements until they are negligible. The algorithm repeatedly applies a complicated similarity transformation to the result of the previous transformation, thereby producing a sequence of matrices that converges to a diagonal form. What is more, the tridiagonal form is preserved." (~~cf. Parlett,~~ [4].) Therefore, the QR algorithm can be regarded as the best sequential algorithm available todate. The question is whether or not the QR algorithm may retain that same effectiveness when mapped into a parallel algorithm on a square or linear multiprocessor array.

In the this note, ~~we shall~~ the author offer an answer to this question using the computational wavefront notion. First, we shall demonstrate that it is advantageous to perform the tridiagonalization of the original matrix by means of a linear array. As the tridiagonalization process requires $O(N^3)$ time on a sequential computing machine, a processing time of $O(N^2)$, using N processing elements in the array, is called for. Secondly, the iteration of the tridiagonal matrix is especially attractive for implementation by means of a linear configuration of the WAP, as the volume of data is linear, i.e. O(2N). Both above operations can conveniently be performed involving local communications only.

---

[1] *Research supported in part by the ONR under contract N00014-81-K-0191 and by the NSF under grant ECS-80-16581.

[2] In signal processing applications, the covariance matrices involved in eigenvalue decomposition are very often symmetric.

The Wavefront Array Processor (WAP)

The wavefront array processor (WAP) [5] is conceived as a programmable variant of the systolic array [6], aimed at solving a majority of matrix algorithms. The topology of most matrix multiplication algorithms can be mapped naturally onto a square or a linear array of processor elements with regular and local interconnections (cf. Fig. 1). To create a smooth data movement in a localized communication network, we make use of the computational wavefront concept. A wavefront in the processing array will correspond to a mathematical recursion in the algorithm. Successive pipelining of the wavefronts will accomplish the computation of all recursions. The pipelining is feasible because the wavefronts of two successive recursions will never intersect (Huygen's wavefront principle), as the processors executing the recursions at any given instant will be different, thus avoiding any contention problems.

The wavefront concept provides a firm theoretical foundation for the design of highly parallel array processors and concurrent languages, and it appears to have some distinct advantages. With respect to the language aspect, the wavefront notion drastically reduces the complexity in the description of parallel algorithms. The mechanism provided for this description is the special purpose, wavefront-oriented language, i.e. the Matrix Data Flow Language (MDFL)[5]. Rather than requiring a program for each processor in the array, this language allows the programmer to address an entire front of processors. As to the architectural aspects, the wavefront notion leads to a wavefront-based architecture which preserves the Huygen's principle, that ensures wavefronts never intersect. Therefore, a wavefront architecture can provide asynchronous waiting capability, and consequently, can cope with timing uncertainties, such as local clocking, random delay in communications and fluctuations of computing-times. In short, the notion lends itself to a (asynchronous) data flow computing structure that conforms well with the constraints of VLSI.

The WAP is, in a sense, an optimal trade-off between the globally synchronized and dedicated systolic arrays [6], (that work on a similar set of algorithms), and the general purpose data-flow multiprocessors. It provides a powerful tool for the high speed execution of a large class of algorithms which have widespread applications. In this note we shall focus on the application of the WAP to the parallel computing of eigenvalue and singular value decomposition problems.

TRIDIAGONALIZATION OF A SYMMETRIC MATRIX

The basic tridiagonalization of a symmetric matrix is implemented by means of the similarity transform:

$$W = Q*A*Q^T$$

where W is tridiagonal and Q is orthonormal. Usually, Q consists of the product of N-1 orthonormal matrices $Q^{(p)}$ such that: $Q = Q^{(N-2)}*Q^{(N-3)}*\ldots*Q^{(2)}*Q^{(1)}$ and $Q^{(p)}$ causes the (N-p-1) lower elements in the pth column of A to be set to zero. Similarly, $[Q^{(p)}]^T$ causes the N-p-1 rightmost elements in the pth row of A to be set to zero. Under the constraint of localized communications, it is preferrable to use a Givens rotation on the matrix for tridiagonalization (rather than, e.g. a Householder transformation). In essence, the operator $Q^{(p)}$, described above, is broken down into a sequence of finer operators, $Q^{(q,p)}$ where each operator annihilates the element a(q,p). Thus, $Q^{(p)}=Q^{(p+2,p)}*Q^{(p+3,p)}*\ldots*Q^{(N,p)}$. Each operator $Q^{(q,p)}$ is of the form:

$$Q = \begin{array}{c|cccccc|c} \text{columns:} & & & q-1 & q & & & \text{rows:} \\ \hline & 1 & & & & & & \\ & & 1 & & & & & \\ & & & \cdot & & & & \\ & & & C(q,p) & S(q,p) & & & q-1 \\ & & & -S(q,p) & C(q,p) & & & q \\ & & & & & 1 & & \\ & & & & & & \cdot\ 1 & \end{array} \qquad (1)$$

$$C(q,p) = \frac{a(q-1,p)}{\left(a(q-1,p)^2 + a(q,p)^2\right)^{\frac{1}{2}}}$$

$$S(q,p) = \frac{a(q,p)}{\left(a(q-1,p)^2 + a(q,p)^2\right)^{\frac{1}{2}}}$$

Of major importance are the following facts: (1) The premultiplication of A by $Q^{(q,p)}$ modifies only rows q-1 and q of A. The elements of those two rows assume the following values after applying the rotation:

$$\begin{bmatrix} a_r'(q-1) \\ a_r'(q) \end{bmatrix} = \begin{bmatrix} C(q,p) & S(q,p) \\ -S(q,p) & C(q,p) \end{bmatrix} \begin{bmatrix} a_r(q-1) \\ a_r(q) \end{bmatrix} \qquad (2)$$

where $a_r'(k)$ represents the row vector containing the elements of row k of matrix A, and $a'(q,p)=0$; (2) The effect of postmultiplying $A_2=(Q*A)$ by Q^T is to modify the elements of columns q-1 and q of A_2 to assume the following values:

$$\begin{bmatrix} a_c''(q-1) & a_c''(q) \end{bmatrix} = \begin{bmatrix} a_c'(q-1) & a_c'(q) \end{bmatrix} \begin{bmatrix} C(q,p) & -S(q,p) \\ S(q,p) & C(q,p) \end{bmatrix} \qquad (3)$$

where $a_c''(k)$ represents column vectors of matrix A_2. As A was symmetric, this operation is largely a repetition of many of the row operations effected in the Q*A process. The exceptions are the four elements located at the junction of rows and columns q and q-1; (3) The sequencing of operations is not quite as rigid and therefore allows for pipelining of wavefronts.

Computational Wavefront

When taking the wavefront viewpoint of the operations, two types of waves are discernable. The first is an advancing wave, related to the row operations up to the diagonal elements and referred to as the "row wavefronts". The second involves computation in the junction regions and the column operations and can be seen as a reflected wavefront along the diagonal. These are dubbed the "column wavefronts".

The wavefront nature can be seen in Fig. 1, which traces the fronts of activity relating to the row operations involved in annihilation of the elements of the first column (row wavefronts). For descriptive purposes, let us temporarily assume a one-to-one mapping of matrix elements onto processing elements, and let each PE include a rotation processor. The wavefront starts at PE(N,1), fetching a(N,2) from above and performing the computation for generating the rotation parameters C(N,1) and S(N,1) which annihilate a(N,1). Upon completing this task, it will further trigger the processor to the right PE(N,2) and the processor above, PE(N-1,1): (1) The rotation parameters will propagate to PE (N,2), and then PE(N,3), etc., each of which will then perform the rotation operations as in eq. (2). (Note that one of the operands is fetched from above, and the updated result will be returned to the PE above.); (2) Almost simultaneously, PE(N-1,1) is triggered to generate its own rotation parameters, and continues to trigger its successor PEs in a similar fashion. In short, the computation activities are propagated upwards and sideways by the first column PEs, and down the rows by all other PEs. Taking a simplified perspective, we can say that the first wavefront activity is started at processing element PE(N,1). PE(N,1) propagates the rotational parameters to PE(N,2) and also triggers the activity of PE(N-1,1), thus forming the second front. They, in turn, activate PE(N,3), PE(N-1,2) and PE(N-2,1) which represent the third front, and so on.

Architecture for Linear Array Eigenvalue Solvers

In order to take advantage of the symmetry of the symmetric eigenvalue problem, let us delete those PEs above the main diagonal, retaining a triangular array. Since the subdiagonal elements are still producing the same results as before, and the superdiagonal elements are simply their transposition, no information will be lost.

We now pose a most imp rtant question: can the square (or triangular) array be utilized with reasonable efficiency in solving the symmetric eigenvalue problem? Our answer to that question is: NO. This critical decision leads us to conclude that, in general, the linear array is the optimal choice.

The reasons supporting this claim can be made clear by a closer examination of the column wavefronts. Fig. 2 shows the sequencing of these column wavefronts and their propagation. (There are several variants of the propagation pattern. This one, however, appears to be the simplest and most representative.) The first column wavefront can be initiated when and only when the first row wave reaches the end of its' travel, i.e. the last two elements of the last two rows. Its' first task corresponds to iterating columns N and N-1 through operator $[Q^{(N,1)}]^T$. The column wave can advance by one stage when the row wave has operated on the last elements of rows N-1 and N-2. In the evolution of the computations, row operations applied to rows p and p-1 must terminate before the corresponding column operations are initiated. This is due to the fact that column operations require data that is the outcome of the row operations. By the same token, the column operations corresponding to annihilation of the (N-p-1) elements of row p (column wave #p) must terminate before the row operations relating to the annihilation of column p+1 (row wavefront #(p+1)) may commence. On the basis of these observations we claim two facts:

1. Unlike the basic QR decomposition problem [11], row wavefront #(p+1) cannot be initiated until the column wavefront #p has reached and updated the values of elements a(N,p+1) and a(N-1,p+1). As each wavefront requires O(N-p+1) time to propagate, and there are (N-2) waves of each kind necessary to annihilate the N-2 columns and rows, the total processing time is $O(N^2)$. Utilization of N^2 PEs in a square array (or even half that number in the triangular array) is extremely inefficient and not cost effective, when compared to the single PE execution time of $O(N^3)$.

2. From Fig. 1 one can also see that essentially at most two PEs in each column are actively executing rotation oriented operations at any time instance. We, therefore, propose to apply the same procedure described above, utilizing a (bi)-linear array of N processing elements. One linear array of processors will implement the row operations, while the other carries out column operations. By the above arguement, we note that the linear array will yield the same $O(N^2)$ execution time as the square (or triangular) array, thus proving that they are unnecessary.

It should be noted that, although the physical configuration of the processor array has changed from square to linear, the nature of the computational wavefront has not, and the theoretical propagation of computational activity is retained. Thus, we have a square array virtual configuration [7] mapped into a linear array actual machine.

Processing Time

It is important to estimate the processing time for the linear array eigenvalue solution. To facilitate the analysis, we shall make a simplifying assumption that each rotation takes one time unit for execution, and that data transfer time is negligible (i.e. zero time units). The critical factor in execution time is the inherent delay between fronts p and p+1 (which eliminate columns p and p+1, respectively). To this end, note that: (1) The first column wavefront can start one time unit after the rotation parameters have been generated (as the parameter transfer time through the array is neglected); (2) The second row front can begin when the first column front has updated the values of a(N,2) and a(N-1,2). This occurs N-1 time units after the generation of the first column front, and N+1 time units after the beginning of the first row wavefront. In general, the pth wave starts N+3-p time units after the (p-1)th wave (for p = 2,..,N-2); thus totalling up to an overall processing time of approximately $N^2/2$.

The scheme presented above has several advantages. First, the final values of the tridiagonal matrix are, upon terminating the procedure, already in their proper placement within the processor array. This allows for pipelining the second phase of eigenvalue determination immediately after the first phase. Thus, once the first and second column annihilation has been completed, PE(1) and PE(2) can commence the activities required by the QR iterations. There is no activity gap between the two execution phases. Secondly, the scheme requires only local communications and is, therefore, well suited for WAP implementation. Finally, each processor may access the elements it processes by stacking them. It can be easily shown that the data elements are in the proper order for this stacking scheme (cf. Fig. 3).

DETERMINING THE EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX

Among the most popular methods for determining the eigenvalue of a symmetric tridiagonal matrix is the iterative diagonalization scheme mentioned above. It uses a series of similarity transformations which retain the symmetricity and bandwidth of the matrix, while reducing the off-diagonal norm and converging to a diagonal matrix, the elements of which are the sought eigenvalues. The algorithm chosen involves repetitive application of the QR algorithm to the matrix A shown in fig. 4, which is the outcome of the first computation phase, that of tridiagonalizing a symmetric matrix.

In the basic QR algorithm, the matrix A is decomposed into the product of an orthonormal matrix, Q, and an upper triangular matrix, R, such that A=Q*R. Thus, $R=Q^T*A$. Postmultiplying R by Q creates $A_2=R*Q=Q^T*A*Q$, so that A_2 is similar to the original A. Rather than generating the decomposing orthnormal matrix Q in a single operation, we choose, as before, to create Q as a product of orthonormal matrices, $Q=Q^{N-1}*...*Q^2*Q^1$, where each $Q^{(p)}$ represents a rotation operator of the type shown in eq. (1), designed to annihilate a single subdiagonal element. The order of application of premultiplications and postmultiplications is flexible. Assume, for the moment, that all premultiplications (row operations) are executed first. The resulting A' is of the form given in Fig. 5. The values of Δ do not have to be computed, as they are redundant. It can clearly be seen that implementation of these iterations involves local dependence only, as the updated diagonal, sub- and super-diagonal values are generated from the original element values in the same and adjacent locations.

$$\begin{matrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot \\ & & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & & b_{n-1} & a_n \end{matrix}$$

Fig. 4: Symmetric Tridiagonal Matrix.

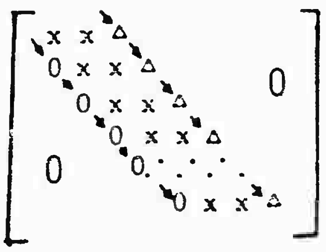


Fig. 5: Matrix of Fig.4 after row modification.

The second phase of the algorithm requires column oriented multiplication which will convert the matrix back to a symmetric, tridiagonal form. The operation involved is similar to that of the row operations described above. Thus the problem is defined by means of an algorithm which adheres to the locality constraint of the WAP. This is, of necessity, the first stage of writing any program for the WAP in a wavefront language, MDFL: define the sequence of operations in such a manner that meets the local communication requirement of the WAP. In most cases, this is done in the most straight forward manner by presenting the algorithm in a matrix-oriented notation, where succession of indices is mapped into geometric adjacency of executing PEs. According to Parlett [4], the diagonalizing processing time is linear.

The following is an MDFL program that implements the tridiagonalization of a symmetric matrix. In order to avoid extensive explanations, we assume the following with regards to the bi-linear array: (1) Every element PE(p) in each row has a FIFO stack of size N-p+1. (2) The elements of each row "push" data onto the stack of the corresponding element in the other row, and "pop" data from their own stack. This, then, replaces the regular FETCH/FLOW data transfer mode between first and second rows. (3) A TOP-OF-STACK pointer is included in each stack. When data is pushed onto the stack, that pointer moves up, and when data is removed from the stack, the pointer travels down. Data cannot be popped from the stack when the TOP-OF-STACK pointer is at the bottom of the stack, thus inducing the wait state inherent to data transfer operations.

```
1   REPEAT
     WHILE WAVEFRONT IN ARRAY DO
       CASE KIND OF
      FIRSTROW: BEGIN                         (*  First-row excludes PE(1).          *)
5                POP A;                       (*  Fetch a(q,p).                      *)
                 FETCH B, LEFT;
                 IF B EQ INF THEN SET TERMINATED;(*  INF represents a flagging value    *)
                                              (*  used to trigger the next wavefront *)
                                              (*  generation.                        *)
                 ELSE
                 BEGIN
10                REPEAT
                    BEGIN
12                  IF NOT TOP-OF-STACK THEN POP B;
                                              (*  Set B=a(q-1,p), otherwise, if at   *)
                                              (*  TOP-OF-STACK, B is set to a(p,p-1). *)
                    FLOW A, RIGHT;            (*  Send a(p,p-  right.                *)
                    FETCH (S, C) LEFT;
15                  FLOW  (S, C) RIGHT;
                    IF TOP-OF-STACK THEN  FLOW (S, C) DOWN;
                    ROTATE (A, B, C, S);
18                  PUSH A;                   (*  Put a'(q,p) on top of lower stack. *)
                                              (*  This operation enables the column  *)
                                              (*  POPs of line 27.                   *)
                    XSR B, A;                 (*  SET A=a'(q-1,p)                    *)
20                UNTIL TOP-OF-STACK;
21                PUSH A;                     (*  Put a'(p,p) on top of stack.       *)
                 END;                         (*  Of ELSE block.                     *)
                 END;

    SECOND ROW  BEGIN                         (*  Column operations.                 *)
25               REPEAT
                    BEGIN
27                  POP A;                    (*  Set A=a'(q,p).                     *
                    IF RIGHT NOT DISABLED THEN
                         BEGIN
30                       FETCH (S, C) UP;
                         FETCH  B, RIGHT;     (*  Get a'(q,p) from right.            *
                         ROTATE (A, B, S, C);
                         FLOW A, LEFT;        (*  Send a''(q,p-1) to PE(p-1).        *
                         FLOW B, RIGHT;       (*  Send a''(q,p) to PE(p).            *
35                       END;
                    FETCH A, LEFT;            (*  Fetch a''(q,p  from left.          *
37                  PUSH A;                   (*  Place a'' (q,p  on top of upper stack *
                                              *  This operation will enable the     *
                                              (*  row wavefront POP operations of    *
                                              *  lines 5, 12, [illegible] and [illegible]  *

                 UNTIL TOP-OF-STACK,
                 END;
40   END CASE,                                *  End of CASE and WHILE blocks.      *
    DECREMENT COUNT,

    UNTIL TERMINATED,
REPEAT 1;
  WHILE WAVEFRONT IN ARRAY DO
  BEGIN
  CASE KIND OF
     CORNER, FIRSTROW:
             BEGIN
             POP A;                       (*  Fetch a(N,p) for generation of      *)
                                          (*  rotation parameters.                *)
             REPEAT
                BEGIN
                POP B;                    (*  Fetch a(q-1,p) from upper stack;    *)
                SINCOS (A, B, C, S);      (*  Generate C and S.                   *)
                FLOW (S, C) RIGHT;
                ROTATE (A, B, C, S);      (*  Now B=a'(q-1,p).                    *)
                TSR B, A;
                END;
             UNTIL TOP-OF-STACK;
             END;
     END;                                 (*  Of CASE block.                      *)
   TSR INF, A;
   FLOW A, RIGHT;
   END;                                   (*  Of WHILE block.                     *)
 DECREMENT COUNT;
UNTIL TERMINATED;
```

LINEAR ARRAY FOR SINGULAR VALUE DECOMPOSITION (SVD)

There are several square array configurations that have been proposed for SVD computations, including those in [8,9]. An alternative to the square array implementation of the SVD employs a linear array which bidiagonalizes the given matrix, A, in emulation of the Golub-Reinsch algorithm [10]. The bidiagonalization procedure is identical, in most respects, to the tridiagonalization routine used above in the symmetric eigenvalue problem.

Once the original matrix A has been transformed into a bidiagonal matrix, the number of data elements active in the processing reduces to 2N-1. The Golub-Reinsch "skipping" sequence is now implemented on these elements by means of Givens rotations. Due to the reduced number of operands, a linear processor array can effectively be used at this stage to converge to the singular values. Thus, the original square matrix is largely impotent and of no use. For this reason it is more beneficial to implement the entire Golub-Reinsch procedure by means of a linear array, in $O(N^2)$ time. The application of the linear array to execution of the diagonalization is, also, very similar to that of the symmetric tridiagonal matrix described above, and will not be further dwelt upon here.

LINEAR OR SQUARE ARRAY?

In this paper we have discussed parallel algorithms for solving eigenvalue and singular value decompositions. Our approach relies heavily on the powerful notion of computational wavefronts and leads to very efficient linear and square array computing structures. In our eigenvalue and singular value decomposing schemes there is strong evidence that a linear array structure can, very often and quite effectively, rival the processing speed achieved by the square array. This, despite the fact that the latter employs considerable more processing elements. The basic facts supporting these claims are: (1) Tridiagonalizing a symmetrical matrix can be performed efficiently in $O(N^2)$ time with a linear array. Thereafter, reducing the tridiagonal matrix can be accomplished in $O(N)$ iterations. (2) Other, non-tridiagonalizing methods, such as RT [8] or modified Hestenes schemes [9] involve full matrix manipulations. These, in general, involve a linear number of iterations of $O(N^2)$ operation each [4]. Thus, a square array of processors can, in general, only achieve $O(N^2)$ processing time, which is the same as the linear array.

REFERENCES

1. R.Schmidt, "Multiple Emitter Location and Signal Parameter Estimation", Proc. RADC Spectral Estimation Workshop Rome, N.Y., 1979, pp. 243-258.

2. G.Bienvenu, "New Principle of Array Processing in Underwater Passive Listening", These Proceedings.

3. N.L.Owsley, "High Resolution Spectrum Analysis by Dominant Mode Enhancement", These Proceedings.

4. B.N.Parlett, "The Symmetric Eigenvalue Problem", Prentice-Hall, 1980.

5. S.Y.Kung, K.S.Arun, R.J.Gal-Ezer and D.V.Bhaskar Rao, "Wavefront Array Processor: Language, Architecture and Applications", Special Issue of the IEEE Trans. Computers on Parallel and distributed Processing, Vol. 31, No. 11, Nov. 1982.

6. C.MEAD and L.Conway, "Introduction to VLSI Systems", Addison Wesley, 1980, Chap. 8.3 by H.T.Kung and C.E.Leiserson.

7. R.J.Douglass, "Algorithm Driven Architecture Design: Algorithms + Alchemy = Architecture", Purdue Workshop on Algorithmically Specialized computer Organizations, Purdue University, West Lafayette, Indiana, Sept. 1982.

8. R. J. Gal-Ezer, "The Wavefront Array Processor: Architecture, Applications and Programming", Ph.D. Dissertation, University of Southern California, Los Angeles, 1982.

9. A. Finn and C. Pottle, "An Algorithm and Simulation Results for a Systolic Array Computation of the Singular Value Decomposition", presented at the SPIE Conf., Arlington, VA., May 1982.

10. G.H.Golub and C.Reinsch, "Singular Value Decomposition and Least Square Solutions", Numer. Math., Vol. 14, pp. 403-420, 1970.

11. S.Y. Kung and R.J.Gal-ezer, "Eigenvalue, Singular Value and Least Square Solvers Via the Wavefront Array Processor", Purdue Workshop on Algorithmically Specialized computer Organizations, Purdue University, West Lafayette, Indiana, Sept. 1982.

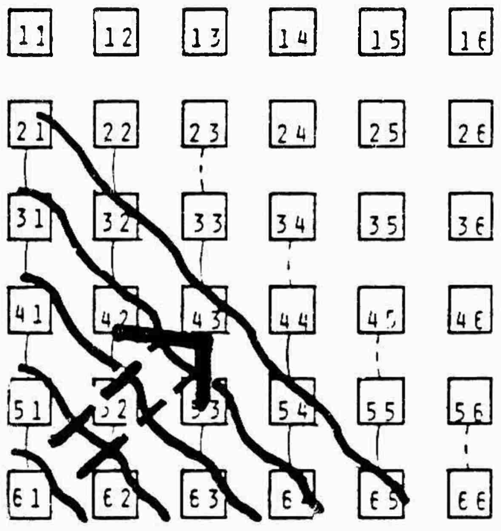


Fig. 1: Row Wavefronts in (virtual) Square Array WAP

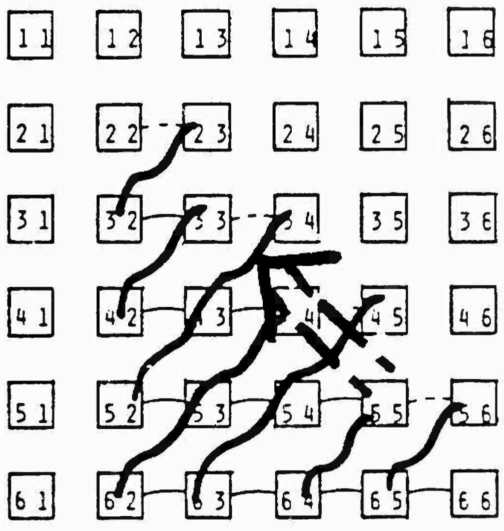


Fig. 2: Column Wavefronts

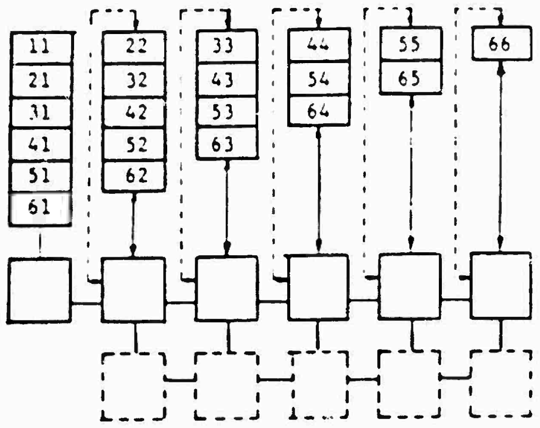


Fig 3: Bi-Linear WAP with Data Stacks for Eigenvalue Decomposition

(Upper row executes row operations, lower row executes column computations.)

SPECIAL PURPOSE ARCHITECTURES: POMISES AND LIMITATIONS[*]

Alan L. Fisher and H.T. Kung
Computer Science Department
Carnegie - Mellon University
Pittsburgh, PA 95213

ABSTRACT

Over the past several years, many special purpose (in particular, systolic) architectures have been proposed as solution to computation bound problems. More recently, implementation and testing of certain of these architectures have begun. In light of these studies, this paper surveys two topics of practical interests. The requisites for successful applications of the special purpose architecture approach, and the outlook for the applications of systolic algorithms.

In order to succeed economically, special purpose design efforts must deal with the following factors:

- Algorithms
- Architectures
- Hardware Design and CAD
- Implementation Technology
- Fabrication Turnaround
- System Integration .

Taking these factors into account, we can assess the promises and limitations of the systolic array approach to signal processing, along with its reductions to practice. Some of the issues we consider are:

- Programmability
- The Design Problem
- Standard Interfaces
- Applications System Development
- Physical Limit on Communication and Synchronization .

[*]This research was supported in part by the Office of Naval Research under Contract N00014 - 76 - C - 0370, NR044 - 422, and N00014 - 80 - C - 0236, NR048 - 659, and in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Airforce Avionics Laboratories under Contract F33615 - 81 - K - 1539; A. Fisher was supported in part by an IBM Graduate Fellowship.

AD P002613

SIGNAL PROCESSING IN HIGH DATA RATE ENVIRONMENTS--DESIGN TRADEOFFS IN THE EXPLOITATION OF PARALLEL ARCHITECTURES AND FAST SYSTEM CLOCK RATES AN OVERVIEW

B. K. Gilbert, T. M. Kinter, D. J. Schwab, B. A. Naused, L. M. Krueger and K. M. Rice
Mayo Foundation
Rochester, Minnesota

F. S. Lee
Rockwell International Microelectronics Research and Development Center
Thousand Oaks, California

Introduction

This conference is exploring the methods by which the emerging very large scale integration (VLSI) technology, i.e., the ability to place more than 10,000 logic gates on a single integrated circuit, can be exploited for the solution of difficult signal processing problems. The following discussion will concentrate on a highly specialized subset of the total signal processing environment, i.e., that small minority of such problems in which a single unprocessed data stream appears at the input of a digital processor in real time and at very high data bandwidths. These high volume data streams must be processed by the "front end" of the signal processor at clock rates equal to or greater than the rates at which they are delivered; in later stages of processing, it may be possible to partition the single high-speed data stream into a series of lower speed substreams, and to institute parallel processing on the substreams. We have been compelled to consider potential solutions to these high data rate problems, and to compare these problems with the capabilities of silicon VLSI, as well as other technologies, with which they may be addressed.

The evolution of digital integrated circuit technology for commercial application, and even to some extent in the military world, has been directed to the fabrication of ever-smaller physical structures and device geometries on mass produced integrated

This work was sponsored in part by contracts F33615-79-C-1875 from the U.S. Air Force, MDA903-82-C-0175 from the Defense Advanced Research Projects Agency, N00014-81-C-2661 from the U.S. Navy, and a research grant from the Fannie E. Rippel Foundation.

circuits. This desire for high device count and high packing density has resulted in an emphasis on structures such as CMOS on bulk silicon and CMOS on sapphire (CMOS/SOS), whose device characteristics permit gate propagation delays in the 2-5 nsec range; the most advanced of these structures can support system clock rates in the 30-50 MHz range. However, several classes of problems, e.g., electronic warfare, radar signal processing, wideband spread-spectrum military communications, and certain biomedical tasks may require unprocessed input data bandwidths as great as .1-2 x 10^9 bytes/second, necessitating system clock rates as high as 2 x 10^9 Hz. High density, low clock rate VLSI technology is simply incapable of solving these types of problems; alternative methods must be identified. One of the most promising of these is the exploitation of "processor speed".

Methods for Achieving Processor Speed

Enhancements in "processor speed" are primarily achieved through the exploitation of faster device technology, i.e., transistors which achieve shorter switching delays. High transistor switching speed is a parameter which is difficult to exploit at the VLSI level, since the attainment of subnanosecond logic gate propagation delays in silicon integrated circuits requires the dissipation of large amounts of power. Power dissipation increases in this manner because large instantaneous current flows are required at a given logic voltage swing to charge the parasitic capacitances of the interconnecting lines between adjacent logical functions on the integrated circuit (or, for that matter, between integrated circuits). If each gate dissipates an average of 1 mW, a 10,000 gate VLSI component would dissipate 10 W, making it difficult to achieve adequate cooling of the integrated circuit in a straightforward manner. Silicon VLSI devices of 10,000 gate complexity should dissipate no more than 100-400 microwatts per gate; at this low power dissipation, switching currents are so small that it is difficult to achieve gate propagation delays much less than 1-2 nsec.

The Exploitation of Gallium Arsenide Digital Device Technology to Obtain Processor Speed

How, then, can those signal processing problems be solved which are simultaneously constrainted by high input data bandwidth and high system clock rate, and which require the design of special components in a low volume production, rapid turnaround environment? One approach currently under intensive investigation in a number of research laboratories involves a synergistic interaction of new technologies which will allow system clock rates above 250 MHz, as well as rapid design and turnaround cycles both at the system and at the component level. Such a capability may be developed by exploiting the physical properties of Gallium Arsenide (GaAs) for the integrated circuit substrate rather than conventional bulk silicon or silicon on sapphire substrates.

The Gallium Arsenide integrated circuit technology has several useful characteristics for such an application. First, the mobility of the electrons in GaAs crystals at a given voltage gradient is six to eight times greater than for silicon, which in turn results in faster electron transit across the channels of N-channel field effect transistors (FETs) fabricated on GaAs substrates. Gate propagation delays can thus be more than three times faster than for similar structures in silicon, or gate power dissipation can be decreased for a given gate speed, or a combination of these two may be achieved.

Gallium Arsenide SSI and MSI integrated circuits, exploiting primarily depletion-mode MESFET SDFL diode-transistor structures, have been successfully fabricated at Rockwell International Microelectronics Research and Development Center, diced, packaged in leaded flat packs or leadless ceramic chip carriers, installed on logic boards, and operated in the Mayo Foundation laboratories at clock rates unachievable with equivalent silicon integrated circuits. Figures 1 and 2 present the performance of single components and of small multicomponent test structures operated over a range of system clock rates from 1-2.5 GHz. It is believed that these data represent the first published demonstration of the performance of multiple interconnected GaAs components operating together in a representative circuit board environment. On-chip gate delays of 80-120 psec, and off-chip risetimes of 250-350 psec, have been measured, with good system noise margins and waveform conformations, even though optimum chip carrier packaging and board design technology were not employed in these early studies. Such speeds are greater than the best achieved in silicon to the present time.

Performance Comparisons of Communications Node Fabricated in Silicon and GaAs

A comparison of present and expected Gallium Arsenide transistor and gate performance characteristics with similar values from high-speed silicon emitter coupled logic (ECL) structures may be enlightening. For example, assuming a silicon-based subnanosecond ECL custom LSI design, we examined the probable performance to be expected from a computer communications network node element with eight input lines and eight output lines. If this node element were fabricated using present state of the art silicon ECL technology, a structure containing approximately 500-700 gates would be required and could be fabricated. The communications node would be able to transmit data from any one or more of the eight input ports to any one or more of the eight output ports at rates of up to 500 megabits/second per line, with a total on-chip power dissipation in the range of four to eight watts. An extrapolation to technology improvements likely to occur by 1985-1987 indicated that advances in lithography techniques would allow this silicon ECL structure to be fabricated with roughly the same number of gates, but exhibiting half the total power dissipation, or twice the throughput (i.e., to a rate of 1 gigabit/second per line), or some combination of decreased power dissipation and enhanced throughput.

The same communications node structure was then designed assuming currently available Gallium Arsenide depletion mode SDFL integrated circuit device technology, using either of two possible designs to achieve the same network node performance. The results from the design exhibiting the lowest gate count are presented here. For single-component gate counts of approximately 500 gates, throughput data rates of 1.5 gigabits/second per line are feasible with 1982 technology at a power dissipation of 1.6 watts, i.e., three times the performance of the 1982 silicon implementation for a power dissipation only 20-40% as high; in a second and more optimistic implementation of this same node, the throughput rates with present GaAs technology would be 3 gigabits/second per line at a power dissipation of .3 watts. The performance achievable in a Gallium Arsenide SDFL implementation in 1985-1987 was estimated, assuming process and device fabrication improvements, to yield transistors with .5-.7 micron transistor channel lengths; throughput rates would then increase to 4 gigabits/second per line at chip power dissipations of .2 watts.

Gate/Macrocell Arrays Fabricated on GaAs Substrates

The speed and power characteristics described above underscore several of the advantages of Gallium Arsenide technology whenever extremely short gate delays or fast system clock rates are a necessity. However, as alluded to earlier, an additional constraint faced by the practitioners of extremely high-speed signal processor design is the small production runs generally undertaken, whether or not the processor incorporates "off the shelf", custom, or semicustom integrated circuits. Recognizing this problem, several laboratories have begun the development of configurable gate arrays and/or configurable macrocell arrays based upon Gallium Arsenide substrates. Mirroring recent results for silicon CMOS configurable gate arrays and configurable macrocell arrays and for first generation silicon ECL macrocell arrays, it appears that Gallium Arsenide gate/macrocell arrays in the 1,000-4,000 equivalent gate size are probably feasible, while even larger gate/macrocell arrays may be feasible. Tradeoffs between the achievement of maximum packing density for custom circuits, and the rapid design cycle possible for the layout and fabrication of semicustom designs on a gate or macrocell array, favor the latter approach if high density is not required or cost ineffective.

Near-Term Technical Feasibility of GaAs-Based Processors

Recent data from a number of sources indicates that it will become technically feasible in the near and intermediate term to assemble and operate Gallium Arsenide processors, or mixed Gallium Arsenide/silicon ECL processors, at clock rates of 250 MHz and above. First, a large number of custom integrated circuits based upon Gallium Arsenide SDFL gate structures have been fabricated and tested (Figures 1 and 2) at small scale and medium scale integrated circuit density levels. Wafer yields have become sufficiently high that these devices are now being included in brassboard versions of processors intended for eventual manufacturing and field installation. Second, the device structures used to implement these custom-designed SSI and MSI GaAs components are at least as easy to manufacture as the equivalent silicon components, although fabrication techniques themselves (based upon ion implantation rather than gaseous diffusion) are quite different. Custom-designed large scale integrated circuits consisting of more than 1,000 equivalent gates have already been successfully tested, although device yields are still unacceptably low. However, both the yields and the device densities on these custom designs appear to be improving in a manner similar to the advances in silicon LSI device densities and yields during the mid and late 1970s.

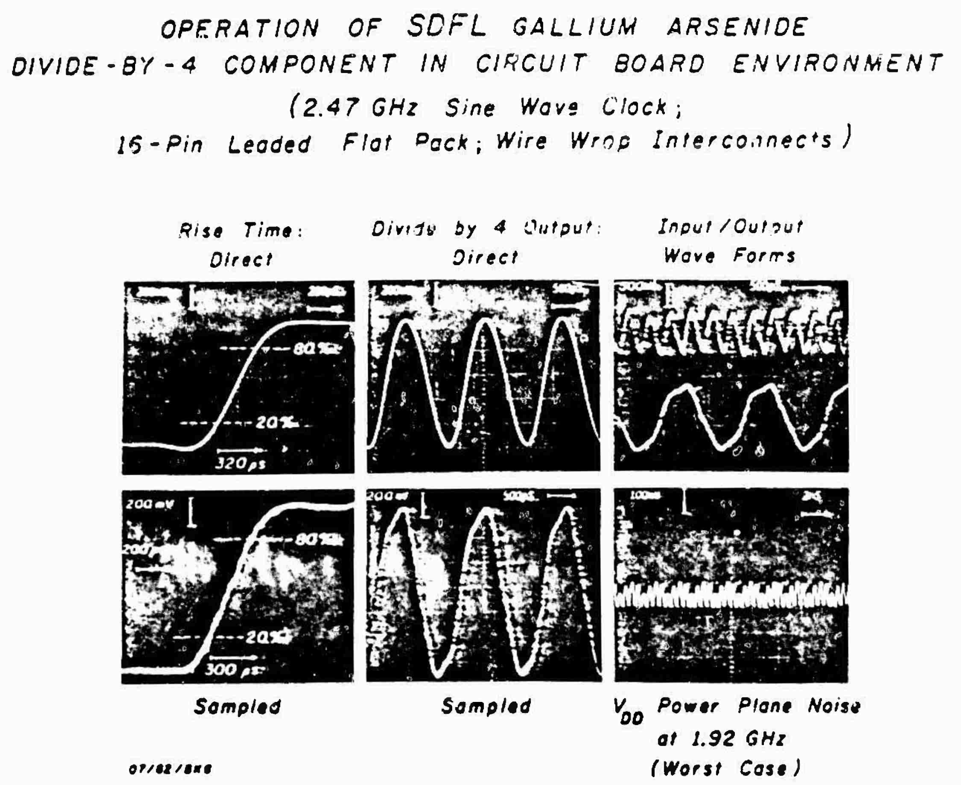
Although there is thus little remaining doubt that custom GaAs components will continue to improve in a number of operational parameters, the same optimism does not prevail among all observers concerning the feasibility of semi-custom structures, i.e., GaAs gate/macrocell arrays. These observers have commented that GaAs gate/macrocell arrays will not be feasible for a variety of detailed technical reasons. However, the first such array, consisting of 100 gates and designed for low power dissipation, has been fabricated and does appear to work properly for small test circuits assembled by interconnection of gates in the array. Gate propagation delays are comparable to the best commercial silicon subnanosecond ECL, even though this structure was optimized for low power rather than for speed. Gate arrays using both bipolar and field effect transistor technology are currently in layout and will undergo testing in 1983. Macrocell arrays are inherently more attractive for high performance devices, particularly on Gallium Arsenide substrates, since any gate structure may be located wherever desired on the integrated circuit to provide minimum spacing between a signal source and its intended destinations. Lastly, much larger gate/macrocell arrays presently appear feasible, which can also benefit from the exploitation of high-speed bipolar structures to maximize transistor fanout capacity and integrated circuit throughput.

This combination of device technologies thus provides a potentially powerful vehicle for the design of very high clock rate, high throughput processors relying upon "custom" integrated circuits, while preserving the option to perform these design and development projects for very small production runs.

In summary, high throughput signal processors based upon Gallium Arsenide integrated circuits operating at system clock rates above 250 MHz will be capable of confronting wide bandwidth signal streams, and will be able to execute algorithms at clock rates equal to or greater than the input data rate. Further advances will be required in Gallium Arsenide chip and substrate technologies, improved packaging for high performance integrated circuits, logic boards which support transmission line interconnects, and rapid-turnaround hierarchical CAD packages. Research is under way in all of these areas; progress to the present has been truly impressive.

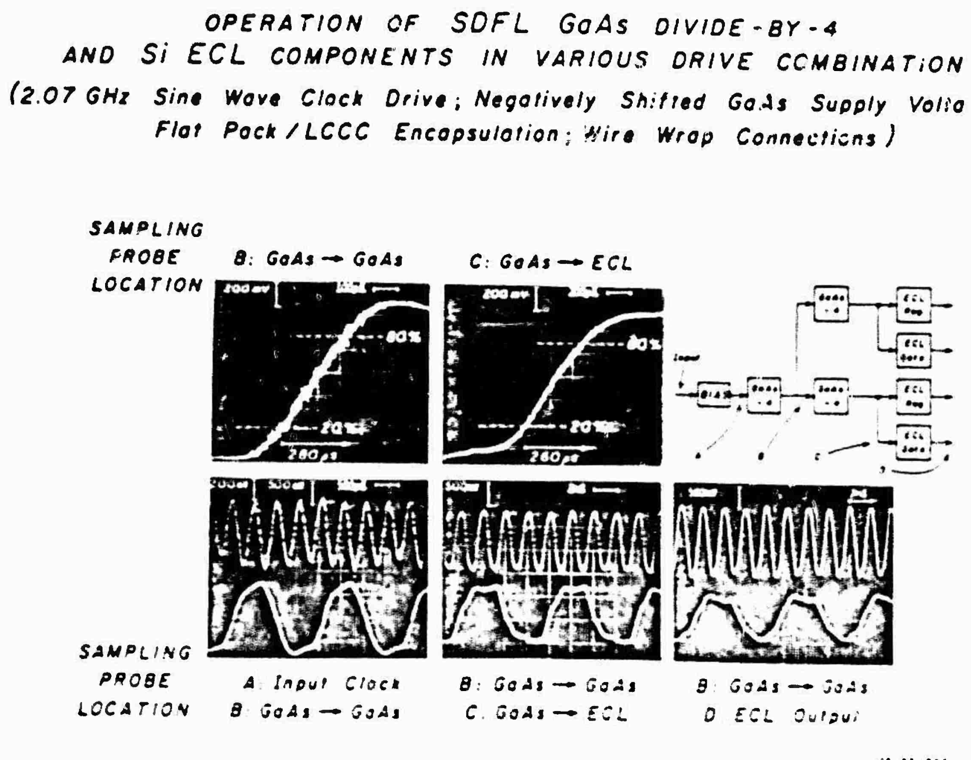
Acknowledgements

The authors wish to thank Mr. A. Firstenberg and Dr. R. Zucca, Rockwell International and Dr. R. C. Eden, Gigabit Logic, Inc., for technical discussions and encouragement; Mrs. E. Doherty, Ms. J. Lauer, and Mr. S. Richardson, for preparation of text and figures.



Operation of a single SDFL Gallium Arsenide divide-by-4 component in a representative circuit board environment. Note similarity of waveform conformations between direct oscilloscope and 14 GHz sampled measurements.

Figure 1



Operation of multiple SDFL Gallium Arsenide and Silicon ECL components interconnected as depicted in diagram. Off-chip rise-times are consistently less than 300 psec.

Figure 2

A VLSI ARITHMETIC PROCESSOR CHIP FOR ARRAY PROCESSING

Roger Wood
University of California
Santa Barbara, California

Glen Culler
CHI Systems, Inc.
Goleta, California

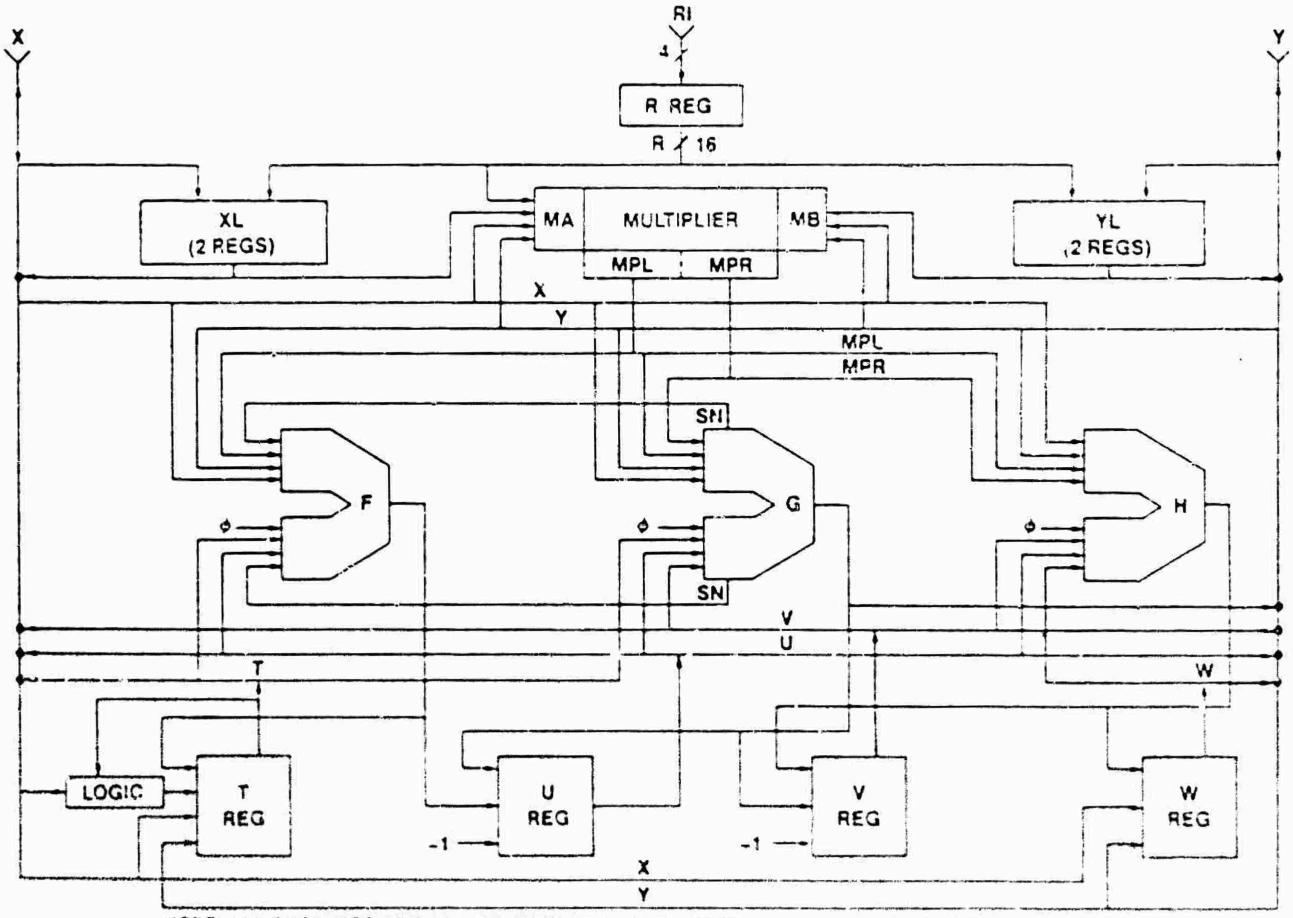
Dave Harrison and Ed Greenwood
Motorola, Inc., Government Electronics Group
Scottsdale, Arizona

ABSTRACT

The described Arithmetic Processor Unit (APU) uses the efforts of Motorola's three micron silicon gate CMOS semiconductor fabrication technology and CHI System's array processor architecture technology. CHI Systems Inc. has developed architectures for array processor systems (as well as the hardware and software elements) for over ten years. In 1979, CHI Systems was developing their fifth iteration of an array processor system, called CHI-5[1]. The U.S. Defense Advanced Research Projects Agency (DARPA) was sponsoring development of the CHI-5. Motorola was seeking to use the CMOS process for low-power and medium-speed linear algebra processors. DARPA proposed the natural combination of these two technologies. Emphasis of this program, sponsored by DARPA, has been to obtain the best deliverable silicon gate CMOS APU capability for a fixed investment. Thus, yield and production cost have been secondary factors. Processing throughput and maximum functions obtainable on the chip were desired features.

The functions on chip, rationale for chip functions, as well as design methodology and benchmarks will be discussed in the full paper.

(1) Patented - U.S. Patent Number 4,287,[illegible] [illegible] Sept. 1, 1981.



NOTE: ALL BUS LINES ARE 16 BITS EXCEPT WHERE NOTED

Figure 1 Basic APU Structure

PARTITIONING BIG MATRICES FOR SMALL SYSTOLIC ARRAYS[*]

Don Heller
Computer Science Dept.
Pennsylvania State University

ABSTRACT

The systolic array concept has proved attractive for design of matrix processors operating in pipelined mode. It may be viewed as a programming technique for parallel computers or as a design technique for special purpose processors. For VLSI implementation, one drawback is the frequent design assymption that the number of input ports exceed some feature size of the input data (rows, columns, nonzero diagonals). We discuss the unfavorable but inevitable situtation where the matrix is too large for the processor array. Partitioning methods are surveyed and their infuence on systolic processor design is considered for some basic matrix computations. The most efficient partitioning methods require use of more than one systolic design, so we argue in favor of programmable processor elements and flexible control of the processor array.

[*] This research is supported in part by the National Science Foundation under Grant [illegible].

"NEW PRINCIPLE OF ARRAY PROCESSING IN UNDERWATER PASSIVE LISTENING"

G. BIENVENU, THOMSON-CSF, Division des Activités Sous-Marines
BP 53 - 06801 CAGNES-SUR-MER CEDEX, FRANCE

H F. MERMOZ, Ingénieur Général des Télécommunications
582, Chemin de la Calade, 83140 SIX-FOURS, FRANCE

1. - INTRODUCTION

One of the main function of an underwater passive listening system is the reckoning of the number of present sources as well as the characteristic parameters of everyone. To do so, the noises transmitted by the sources are used when recorded on the sensors of an array. The basic tool is the spatial processing. The traditional tool is the classical beamforming. Then, in view of improving the performance, one came to adaptive beamforming of a sensor array. As a result, this method brings an improvement -an array gain- which is asymptotically bound by the signal to noise ratio of the source noises, when measured on a particular sensor. More recently have appeared more powerful methods called "high resolution". The improvement in performance as compared to previous processing is at the cost of one more assumption on the medium. Nevertheless, these methods carry the possibility to include free parameters in the medium model, and that make more flexible the assumptions to be accepted.

Adaptive array processing, as conventional beamforming, needs hypotheses only on the sources : they are pointlike, perfectly spatially coherent, and the wavefront shape from a source is a known function of the source position (the transfer function of the sensors is also supposed known). The additional hypothesis needed by high resolution methods concerns the background noise : its spatial coherence has to be known. High resolution methods have better performance than those of adaptive array processing thanks to that hypothesis. They suppose also that the number N of sources is less than the number K of sensors.

2. - BASIC PRINCIPLES OF HIGH RESOLUTION METHODS

The basic hypothesis made on the background noise is that it is statistically independent between the sensors. Therefore, the cross spectral density matrix (CSDM) $\Gamma(f)$ of the received signals is written according to the hypotheses made :

$$\Gamma(f) = \sigma(f)\, I + \sum_{i=1}^{N<K} \gamma_i(f)\, \vec{D}_i(f)\, \vec{D}_i^{+}(f) = \sigma(f)\, I + \Gamma_s(f) \qquad (1)$$

$\Gamma(f)$ is a (KXK) matrix
$\sigma(f)$ is the spectral density of the background noise
I is the spatial coherence matrix of the background noise that is equal to the identity matrix
$\gamma_i(f)$ is the spectral density of source i
$\vec{D}_i(f)$ is the source position vector composed of the normalized transfer functions between source i and the K sensors.

High resolution methods are based on the eigenvector-eigenvalue decomposition of $\Gamma(f)$. It can be shown that :

- (K-N) eigenvectors are equal to $\sigma(f)$, and the N others are greater
- the N eigenvectors $\vec{V}_i(f)$ ($i \in [1,N]$) related to the N maximum eigenvalues $\lambda_i(f)$ are in the N dimensional subspace spanned by the N source position vectors called the source subspace. An important relation is :

$$\Gamma_s(f) = \sum_{i=1}^{N} \gamma_i(f)\ \vec{D}_i(f)\ \vec{D}_i^{+}(f) = \sum_{i=1}^{N} [\lambda_i(f)-\sigma(f)]\ \vec{V}_i(f)\ \vec{V}_i^{+}(f) \qquad (2)$$

- the (K-N) eigenvectors $\vec{V}_{io}(f)$ ($i \in [N+1,K]$) related to the (K-N) minimum and equal eigenvalues are orthogonal to each of the N source position vectors. They spanned a (K-N) dimensional subspace called the orthogonal subspace.

This analysis yields the principle of high resolution methods :

a) From the eigenvalues of $\Gamma(f)$ are deduced the number of sources (which is the number of sensors minus the number of equal and smallest eigenvalues), and the source subspace and the orthogonal subspace.

b) The source locations can be obtained using either the source subspace, or the orthogonal subspace. In both cases, a model for the source position vector $\vec{D}(f,\vec{\theta})$ has to be used ($\vec{\theta}$ stands for the source position parameters). With the source subspace, the source parameters $\vec{\theta}_i$ and $\gamma_i(f)$ can be looked for, such as the following equation is satisfied [1,2]

$$\sum_{i=1}^{N} [\lambda_i(f)-\sigma(f)]\ \vec{V}_i(f)\ \vec{V}_i^{+}(f) = \sum_{i=1}^{N} \gamma_i(f)\ \vec{D}(f,\vec{\theta}_i)\ \vec{D}^{+}(f,\vec{\theta}_i) \qquad (3)$$

Modified maximum likelihood or maximum entropy methods can also be used [3]. With the orthogonal subspace [4,5], the source positions are given by the values of $\vec{\theta}$ for which the following function comes to a null :

$$G(\vec{\theta}) = \sum_{i=N+1}^{K} |\vec{V}_{io}^{+}(f)\ \vec{D}(f,\vec{\theta})|^2 \qquad (4)$$

The above properties are of course asymptotical ones, since, in practice, only an estimation $\hat{\Gamma}(f)$ of $\Gamma(f)$ can be obtained. Thus only estimates of the sources parameters can be gotten. The (K-N) smallest eigenvalues are not, in practice, strictly equal. Therefore, the decision about the number of sources and equivalently the source subspace and the orthogonal subspace, is relevant of the detection theory. It has been shown [6] that it is possible to find a test for the number of sources which depends only on the eigenvalues of $\hat{\Gamma}(f)$. It has been shown simultaneously that the properties of the eigensystem of $\Gamma(f)$ remain valid for the eigensystem of $\hat{\Gamma}(f)$.

3. - GENERALIZATION OF HIGH RESOLUTION METHODS

The main property of high resolution methods is that their resolution power is no longer limited by the signal to noise ratio as for adaptive processing, but it increases with the observation time up to infinity. Therefore, asymptotically, two sources can be resolved how close and weak they may be. Physical limitations come from the non perfect knowledge and the fluctuations of both the spatial coherence of the background noise and the shape of the source wavefronts. But it can be shown that the assumptions about these two quantities can be made more flexible, which increases the validity domain of high resolution methods.

3.1. Modeling of background noise spatial coherence [7]

At sea, the background noise is somewhat correlated between the sensors. Its spatial coherence matrix is not an identity matrix. But if it can be modelized as a known function of unknown parameters $\vec{m}$: $J(f,\vec{m})$, the unknown parameters can be determined. For a given $\vec{m}$, there exists a known matrix $C(f,\vec{m})$ such that :

$$C(f,\vec{m})\ J(f,\vec{m})\ C^{+}(f,\vec{m}) = I \qquad (5)$$

Let $\vec{m}_o$ be the value of the background noise parameters as it is at sea. Consider the matrix : $\Gamma_c(f,\vec{m}) = C(f,\vec{m})\ \Gamma(f)\ C^{+}(f,\vec{m})$.
Thus : $\Gamma_c(f,\vec{m}) = \sigma(f)\ C(f,\vec{m})\ J(f,\vec{m}_o)\ C^{+}(f,\vec{m}) + \sum_{i=1}^{N} \gamma_j(f)\ \vec{D}_{ci}(f,\vec{m})\ \vec{D}_{ci}^{+}(f,\vec{m})$
Where : $\vec{D}_{ci}(f,\vec{m}) = C(f,\vec{m})\ \vec{D}_i(f)$.
It is clear that if $\vec{m}$ is moved through different values, when it reaches $\vec{m}_o$, $C(f,\vec{m})\ J(f,\vec{m}_o)\ C^{+}(f,\vec{m})$ becomes equal to the identity matrix. Thus if the eigenvalues of $\Gamma_c(f,\vec{m})$ are plotted versus $\vec{m}$, when $\vec{m} = \vec{m}_o$, the (K-N) eigenvalues come to be equal, property which gives the value of $\vec{m}_o$. Knowing $\vec{m}_o$, the source subspace and the orthogonal subspace can be determined.

Of course in practice, only an estimate $\hat{\Gamma}(f)$ of $\Gamma(f)$ is available. Thus only a focusing point is observed versus $\vec{m}$ for the eigenvalues of : $\hat{\Gamma}_c(f,\vec{m}) = C(f,\vec{m})\ \hat{\Gamma}(f)\ C^{+}(f,\vec{m})$, and thus only an estimate of $\vec{m}_o$ is obtained.

3.2. Exploitation of the source subspace

As for the background noise spatial coherence, less stringent assumptions on the wavefronts of the sources can be accepted.

If nothing is known about the wavefronts, the source position vectors can be deduced from the reconstructed spectral density matrix of the source alone (relation (2)) but only as functions of N(N-1)/2 unknown scalar parameters. Thus some a priori knowledge is needed in order to have a model which yields another expression of the source position vectors. Therefore, the identification of the two expressionsof the N source position vectors, every one with K components, leads to KN equations. As the number of unknowns are the N(N-1)/2 preceding parameters, and the N spectral densities and the 3N coordinates of the sources, the model of the source position vector can include Z "medium descritive" parameters in order to have as many unknowns as equations. Thus :

$$Z = N\left[K - \frac{N+7}{2}\right]$$

Therefore the medium is all the better described than the number of exploring sensors and of sources which "light" is larger.

In the above method, the source parameters are directly obtained. Another procedure may be proposed which is based on the feeling that the am at of a priori to be accepted in order to express the source vectors themselve is much "lighter" and more credible than the amount needed for a model of the medium. In this method, a source vector model which does not include the source position is used first to determine the wavefronts of the sources, and the positions are obtained in a second step.

It is shown than propagation parameters can also be included in the model with the orthogonal subspace method.

4. - HIGH RESOLUTION METHOD IMPLEMENTATION

The general structure of the processing scheme can be devided in three main parts :

- Input data preprocessing : in this part the FFT of the received signals is performed, and after that there are L identical channels,one for each frequency cell . The second step of the preprocessing is cross spectral density matrix estimation : the computations are complex matrix multiplications and additions.

- High resolution methods specific calculations : in this part, the parameters of the background noise spatial coherence are estimated, and then the source subspace and the orthogonal subspace are determined. The calculations needed are of a new sort in array processing and typically belong to the matrix computation domain. They consist in eigensystem decomposition of the cross spectral density matrices.
- Source parameter estimation : in this part, the source parameters are estimated using the source subspace or the orthogonal subspace. The complexity of the calculations depends on the number of propagation parameters. They are mainly scalar products of complex vectors or resolution of systems of equations, but which are generally not linear.

5. - CONCLUSION

High resolution methods are very interesting because of their improved resolving capability compared to that of conventional processing. They leads to new types of architectures and of computations. The calculations mainly belong to the field of matrix operations. Thus new parallel computation methods as systolic array processing must be an appropriate solution [8].

- ACKNOWLEDGEMENTS

The work presented here bas been in part supported by the Direction des Recherches Etudes et Techniques, Paris, France.

- REFERENCES

[1] - LIGETT W.S. "Passive sonar : fitting models to multiple time series" Proceed. of NATO ASI on Signal Processing, Loughborough, (U.K.), 1972, pp. 327-345.

[2] - MERMOZ H. "Imagerie, corrélation et modèles" Ann. des Télécom., t. 31, n°1-2, Jan. Feb. 1976, pp. 17-36.

[3] - OWSLEY N.L., LAW J.F. "Dominant mode power spectrum estimation" Proceed. ICASSP 82, Paris, France, May 3-5, 1982, pp. 775-778.

[4] - BIENVENU G. "Influence of the spatial coherence of the background noise on high resolution passive methods" Proceedings of ICASSP 79, Washington D.C. 2-4 April 1979, pp. 306-309.

[5] - SCHMIDT R. "Multiple emitter location and signal parameter estimation" Proceed. RADC Spectrum Estimation Workshop, Oct. 1979, pp. 243-258.

[6] - BIENVENU G., KOPP L. "Optimality of high resolution array processing using eigensystem" Submitted to ASSP Proceedings, feb. 1982.

[7] - BIENVENU G., KOPP L. "Adaptivity to background noise spatial coherence for high resolution passive methods" Proceedings of ICASSP 80, Denver, Co. 9-11 April 1980, pp. 307-310.

[8] - BROMLEY K., SYMANSKI J.J., SPEISER J.M., WHITEHOUSE J.H. "Systolic array processor developments" Proceed. of the CMU Conf. on VLSI Systems and Computations, Carnegie-Mellon University, Pittsburgh, PA., 19-21 oct. 1981.

HIGH RESOLUTION SPECTRUM ANALYSIS BY DOMINANT MODE ENHANCEMENT

Norman L. Owsley
Naval Underwater Systems Center
New London, Connecticut 06320

ABSTRACT
(Extended Summary)

(5,6)

High-resolution, data adaptive spectrum analysis techniques can exploit prior knowledge of the signal dimensionality and noise coherence to obtain even higher resolution capabilities than previously envisioned (Lacoss). The classical minimum variance (maximum likelihood) and maximum entropy spectrum estimators form the basis for corresponding enhanced estimators which require only a prior knowledge of signal bandwidth to provide resolution which is limited only by observation time. The basic idea is to identify only that portion of the estimated data covariance matrix which corresponds to the coherent signal portion of the analysis data. This can be accomplished alternatively by eigensystem analysis, singular value decomposition and Cholesky factorization of the estimated "signal only" covariance matrix. The resultant signal covariance matrix decomposition is formally substituted into the high-resolution spectral estimator along with a scalar multiplier called an enhancement factor. As the enhancement factor is made large, the ability of the analysis process to separate closely spaced spectral components increases until it is limited only by the amount of averaging time available to estimate only the largest eigenvalues and corresponding eigenvectors. The enhanced maximum entropy appears to have better resolution capabilities than the enhanced minimum variance. This is a benefit only when two spectral lines have to be resolved to reduce the component of frequency estimator total RMS error due to bias. Otherwise, the enhanced minimum variance estimator has less RMS error due to random fluctuation which is an advantage when only moderately enhanced resolution is required. The new and important problem of estimating the range reponse of a linear array of passive sensors is presented as an example which illustrates all of the important aspects of the topic.

INTRODUCTION

High-resolution power sectrum analysis methods have well known applications to single channel frequency analysis (1) and multiple channel sensor array frequency-wavenumber analysis (2). More recently, sensor array simultaneous frequency-wavenumber-range analysis has provided a further extension of fundamentally the same power spectrum analysis procedures (3). Principal among these techniques are, first, the minimum variance, distortionless response (MV) filter which is closely related to maximum likelihood procedure (1,2) and, secondly, the class of last-squares linear smoothing/prediction techniques referred to herein corporately as the maximum entropy (ME) method (5,6,7). It is typical of these techniques to assume no prior knowledge of the vector space dimensionality of either the signal or noise components of the data to be analyzed. Such prior knowledge of the signal takes the form of total bandwidth, angular and radial extent of the signal for frequency, wavenumber and range analyses respectively. Prior knowledge that the noise can be prewhitened and the signal is known to have a dimensionality which is small relative to the analysis data dimensionality can be exploited to enhance the spectrum estimator resolution capabilities to an extent limited only by the observation time. Accordingly, an approach to data adaptive spectral analysis is described herein which can be characterized by a modal decomposition of the observation data covariance matrix. This approach, which can be expressed alternatively in terms of either orthonormal (8,9,10), singular value (10) or Cholesky decompositions of the covariance matrix, is in contrast to either the Gram-Schmidt orthogonalization procedures (11) or related schemes as embodied in the lattice filter structure (12).

The use of eigenvector orthonormal decompositions for frequency-wavenumber spectrum analysis has evolved from nonparametric and adaptive array processing schemes (13,14,15) to applications as high-resolution spectral estimators (9,10,15-22). Frequently, these high resolution techniques are viewed as ad hoc methods based on the notion of separability of signal and noise processes into orthogonal vector subspaces. In fact, two distinct versions of these high resolution spectrum estimators are derived herein from the formal expressions for the minimum variance, distortionless response (MV) and maximum entropy method (ME) spectrum estimators. The additional prior information which allows the extension of the MV and ME estimators to their so-called enhanced high resolution, dominant mode forms is either that the noise is uncorrelated or can be prewhitened and the signal components of the analyzed data are spectrally narrow.

In this paper, the dominant mode form of the analysis data covariance matrix is presented along with the expression for the readily derived inverse of this matrix. Next, the ~~MV~~ and ME spectrum estimators in modal decomposition form are developed along with 3 dB down resolution expressions. Finally, the modal MV and ME estimators are applied to passive sensor array processing for source range estimation.

SIGNAL MODEL

Let the covariance matrix for the stationary, zero mean complex data N-vector $\underline{x}(t)$ at discrete time t be given by the expectation

$$\underset{\sim}{R} = E\left\{ \underline{x}(t)\ \underline{x}(t)' \right\} \qquad (1)$$

where the notation ' indicates the complex conjugate transpose. Of course, in practice, a time averaged estimate of $\underset{\sim}{R}$ can be used instead of $\underset{\sim}{R}$ in terms of the ensemble average as defined in Eq. (1). This covariance matrix is assumed to consist of the signal and noise components,

$$\underset{\sim}{R} = \underset{\sim}{P} + \sigma^2 \underset{\sim}{I}_N$$

where $\underline{P}$ is a rank $K \leq L << N$ signal covariance matrix, σ^2 is the uncorrelated noise variance and $\underline{I}_N$ is an N-by-N unit diagonal matrix.

In terms of a dominant mode(s) decomposition we have

$$\underset{\sim}{P} = \sum_{k=1}^{K} \lambda_k \underline{M}_k \underline{M}_k' = \underset{\sim}{M}\ \underset{\sim}{\lambda}\ \underset{\sim}{M}'$$

where $\underset{\sim}{M}$ denotes an N-by-K modal matrix of P, and $\underset{\sim}{\lambda}$ denotes the diagonal matrix formed by the eigenvalues of P. An enhanced data covariance matrix can be written as

$$\underset{\sim}{R}(e) = e\ \underset{\sim}{M}\ \underset{\sim}{\lambda}\ \underset{\sim}{M}' + \sigma^2 \underset{\sim}{I}_N \qquad (2)$$

where the scalar parameter e is referred to as the modal enhancement factor.

SPECTRUM ANALYSIS

Two approaches to high resolution spectrum analysis are presented in the following. Both methods result from a linearly constrained quadratic minimization problem formulation in conjunction with the additional information that, first, the signal vector subspace is of dimension K, second, the noise process is zero mean, independent and identically distributed and, third, the spectral resolution is to be maximized. The maximization of resolution is equivalent to saying that the enhancement factor, e, is to be made large. This is because a large e is equivalent to a high signal-to-noise ratio which is the principal factor in resolution improvement.

Minimum Variance Distortionless Response (MV)

For this procedure, it is desired to find a linear filter vector $\underline{W}$ for the analysis data vector which is a solution to the following constrained minimization problem.

Minimize: $\sigma^2_{MV} = E\left\{\left|\underline{W}'\ \underline{x}(t)\right|^2 / e\right\}$

$= \underline{W}'\ \underset{\sim}{R}(e)\underline{W}$

Maximize: Spectral resolution, i.e., $e \rightarrow \infty$

Constraints: (a) Distortionless (unit) response requires

$$1 = \underline{W}'\underline{D}(\theta)\ .$$

(b) Signal space of dimension K and uniform independent noise requires Eq. (2).

Maximum Entropy (ME)

For smoothing and prediction it is desired to find a filter vector $\underline{A}$ which is to be applied linearly to the data vector $\underline{x}(t)$ in such a way that the n-th element in $\underline{x}(t)$, say $x_n(t)$, is estimated in a least squares sense by a linear combination of the other N-1 elements of $\underline{x}(t)$. This objective in conjunction with both resolution maximization and the a priori knowledge that the data covariance matrix has the structure specified by Eq. (2) can be stated as the following constrained optimization problem.

Minimize: $\sigma^2_{ME} = E\left\{\left|\underline{W}'\underline{x}(t)\right|^{2/e}\right\}$

Maximize: spectral resolution, i.e., $e \rightarrow \infty$

Constraints: (a) Smoothing/Prediction at point n in data window requires

$$1 = \underline{W}^{+}\underline{1}_n$$

where $\underline{1}_n$ is a real N-vector consisting of all zeroes except a one for the n-th element (b) Eq. (10).

(b) Signal space of dimensionality K and uniform independent noise requires Eq. (10).

In the full paper, the solutions to both of the above problems, will be examined. Other issues including resolution performance, computation of the enhanced spectrum estimators, and high resolution range estimation will also be discussed.

CONCLUSIONS

Two high resolution spectral analysis procedures based on the well-known minimum variance (MV) and maximum entropy (ME) have been presented. These procedures exhibit superior resolution performance relative to either the MV or ME spectrum estimators with ultimate performance limited only by observation time. Such performance should have substantial impact in spectral analysis applications which are estimator bias limited as opposed to estimator random fluctuation limited.

REFERENCES

1. Lacoss, R. T., "Data adaptive spectrum analysis methods," Geophysics, Vol. 36 No. 4, August 1971, pp. 661-675.

2. Capon, J., "High-resolution frequency-wavenumber spectrum analysis," Proc. IEEE, Vol. 57, 1969, pp. 1408-1418.

3. Owsley, N. L. and G. Swope, "High resolution range estimation with a linear array," Proceedings of IEEE EASCON '82, Washington, D.C., September 1982.

4. Owsley, N. L., "An overview of optimum adaptive control in sonar array processing" in Applications of Adaptive Control, Ed. K. S. Narendra and R. V. Monopoli, Academic Press, 1980, pp. 131-164.

5. Burg, J. P., "Maximum entropy spectrum analysis," presented at the 37th Annu. Meeting Soc. Exploration Geophysicists, Oklahoma City, Okla., 1967.

6. Burg, J. P., "The relationship between maximum entropy and maximum likelihood spectra," Geophysics, Vol. 37, No. 2, April 1972, pp. 375-376.

7. Vanden Bos, A., "Alternative interpretation of maximum entropy spectral analysis," IEEE Trans. Inform. Theory (Corresp.), Vol. IT 17, July 1971, pp. 493-494.

8. Owsley, N. L., "Adaptive data orthogonalization," Proc. IEEE ICASSP, Tulsa, Okla., April 1978, pp. 109-112.

9. Owsley, N. L., "Modal decomposition of data adaptive spectral estimates," Yale University Workshop on Applications of Adaptive Systems Theory, Ed. K. S. Narendra, May 1981.

10. Owsley, N. L. and J. F. Law, "Dominant mode power spectrum estimation," Proc. IEEE ICASSP, Paris France, April 1982.

11. Tufts, D. and R. Kumaresen, "Data-adaptive principal component signal processing," Proc. 1980 IEEE CDC, Alberquerque, N.M., December 1980.

12. Griffitns, L. J., "A continuously adaptive filter implemented as a lattice structure," Proc. IEEE ICASSP, Hartford, CT, 1977.

13. Liggett, W. S., "Passive sonar: fitting models to multiple time series," Signal Processing, Ed. J.W.R. Griffiths, et al, Academic Press, 1973.

14. Owsley, N. L., "A recent trend in adaptive spatial processing for sensor arrays: constrained adaptation," Signal Processing, Ed. J.W.R. Griffiths, et al, Academic Press, 1973.

15. Mermoz, H., "Complementarity of propagation model design with array processing," Aspects of Signal Processing, Ed. G. Tacconi, Reidel, 1977.

16. Cantoni, A. and L. Godara, "Resolving the directions of sources in a correlated signal field incident on an array," Journ. of Acoust. Soc. Ameri., 67(4), April 1980, pp. 1247-1255.

17. Bienvenu, G. and L. Kopp, "Adaptive high resolution spatial discrimination of passive sources," Underwater Acoustics and Signal Processing, Ed. L. Bjorno, D. Reidel, 1981.

18. Klemm, R., "High-resolution analysis of nonstationary data ensembles," in KUNT and COULON, Signal Processing: Theories and Applications, October 1980, pp. 711-714.

19. International Mathematical and Statistical Libraries Inc., Eigensystem Analysis, Vol. 2, Chap. E, Edition 9, 1982.

20. Tufts, D. and R. Kumarasen, "Singular value decomposition and spectral analysis," Proc. of IEEE ASSP Workshop on Spectral Analysis McMaster Univ., Hamilton, Ont., August 1981.

21. Johnson, D., "Improving the resolution of bearing in passive sonar arrays by eigenvalue analysis," Proc. First ASSP workshop on Spectral Analysis, McMaster University, August 1981.

22. Gabriel, W., "Adaptive superresolution of coherent RF spatial sources," First ASSP workshop on Spectral Analysis, McMaster University, August 1981.

AD P002616

State of the Art in Eigenvalue Computations

B. N. Parlett[†]

Extended Summary

The subject must be divided up according to two obvious but unappreciated criteria. Matrices are either *large* or *small* (see Section 3) and their eigenvalues are either *real* or *complex*. Moreover, the property that produces real eigenvalues in the overwhelming number of cases is the *symmetry* of a real matrix.

Definition: An $n \times n$ matrix is *small* (in a given computer system) if it and its matrix of eigenvectors can be held as conventional square arrays in the fast (random access) store or memory. Otherwise, it is *large*.

This is a useful distinction. There is a sharp difference between the techniques used for the two cases (large and small), and also between the very problems which give rise to the matrices. On conventional number crunching systems (CDC 7600, 6600; IBM 370/195, 165) a 100×100 matrix is small. On a small mini-computer system a 50×50 might well be large. Nevertheless we may safely say that 1000×1000 is large and we must note that quantum physicists would like to compute the spectra of matrices of size 2^p with p rising to 20, 30, 40 and beyond!

Small Matrices

Most numerical analysts consider that the eigenvalue problem is solved for small matrices. Results are as accurate as the stability of the problem and the precision of the computer permit.

There is one open problem of a theoretical nature (convergence of shifted QR for nonnormal matrices) and I am not aware of any computational bottlenecks, complaints, or unfulfilled requests in this category

† Department of Mathematics, and the Computer Science Division of the Department of Electrical Engineering and Computer Science, University of California at Berkeley

The work of the author was supported in part by the Office of Naval Research under contract N00014-76-C-0013.

It would be interesting to learn of important computing tasks for which the current quality of performance is not adequate. Real time applications come to mind.

Another aspect to bear in mind is that it is just not worth trying to exploit special patterns of zero elements in *small* matrices (their so-called sparsity structure). The only exception is when the matrix has very narrow bandwidth (e.g. tridiagonal, pentadiagonal). In other words there is enough overkill in the techniques used for *small* matrices that there is no point in exploiting any special properties of the matrix (except for tridiagonality).

Another fact which surprises many people is that it is as easy (i.e. as quick) to compute *all* the eigenvalues of a small, full, real symmetric matrix (with $n \geq 30$) as to multiply that symmetric matrix by another one! Briefly, "eigenvalues are easier than multiplication".

Suppose that we drop the adjective symmetric in the previous paragraph. The assertion is then false, but not by much. Of course, the eigenvalues may well be complex. Nevertheless, they can *all* be computed in less time than it takes to form five matrix-matrix products, i.e. $5n^3$ scalar multiplications. Again, n must be large enough so that its leading powers dominate the others, say n = 20, but still be small.

This state of affairs would have been unbelievable in 1955. What made the difference?

The programs which achieve these attractive performance levels are widely available through libraries such as EISPACK, IMSL, or NAG, which are well documented and have been savagely tested. They embody what was learnt between 1958 and 1970 by the small cooperative group of experts.

Large Matrices

The picture changes sharply when the matrix (the input) and the results (the output) can no longer be held in the random access memory. Such problems are vital to a variety of users. Some good methods have been developed, research is active, but there is nothing like EISPACK available. Indeed, some of the good methods are buried inside special purpose packages (e.g. NASTRAN, ADINA for structural analysis, and the Quantum Chemistry Program Exchange at Indiana).

Two aspects of the problem need emphasis.

(1) It is important to exploit any *sparsity structure* that the original problem may enjoy.

(2) The efficiency of a technique depends quite strongly on the *computer system* (virtual memory or explicit transfer between primary and secondary storage).

The information demanded from large problems is less varied than for small. The symmetric case dominates. I have never heard of a need for the complete spectral factorization (all eigenvalues and all eigenvectors) but I do know of one application using the whole spectrum! The dominant demand (by far) is for some eigenvalues (maybe 3, maybe 50) at the left end of the spectrum (near 0) together with the associated eigenvectors. In 1978 an engineering company spent $12,000 of computer time for 30 eigenpairs of a problem with $n = 12{,}000$. This figure excludes the cost of program development.

There is a beautiful way to exploit sparsity without developing a different method for every important pattern of zero elements. The *user* supplies a program which takes any n-vector v as input and produces the n-vector $u = Av$ as output. The method must then compute eigenvalues and eigenvectors by supplying suitable vectors v to the program. The linear operator A is never known to the solution method. No transformations on A are possible. In this way the burden is on the user to exploit what is known about A to make the computation of u ($=Av$) as efficient as possible. In practice A will not be a simple matrix. A typical situation involves a diagonal matrix D, a structured matrix K, and a shift parameter σ. In fact $A = D(K-\sigma D^2)^{-1}D$. No inversion please, just solve linear systems, however painful that may be. The output u is found in three steps:

(i) form $w = Dv$, (ii) solve $(K-\sigma D^2)x = w$ for x, (iii) form $u = Dx$.

Please note that step (ii) itself might employ some iterative method for solving the system whenever it is inconvenient to factor $K-\sigma D^2$. For big problems it is sometimes necessary to use secondary storage for step (ii).

It is not surprising that vector computers (particularly the Cray-1 and Cyber 205) are attractive devices for problems where the vectors are long and full but the matrices are sparse.

Nor is it surprising that the methods which have emerged as champions for serial computers exploit heavily the characteristics of these machines: ready access by the processor to a large store, facility for complicated nesting of loops in programs. It would be surprising if such methods were well suited for parallel processors.

Parallel Computation

It seems that research on parallel algorithms is focused on surpassing serial machines on standard *small* problems ($n \leq 100$) such as matrix-matrix multiplication or solution of linear equations. Yet these same tasks are ignored by the conventional matrix experts who feel that nearly all needs are being satisfactorily met for small problems. Their research efforts are directed to large problems, either perfecting general methods or devising special techniques for important special cases, such as solving quadratic λ-matrices, $(\lambda^2 A + \lambda B + C)x = 0$, or use of the Cray-1 in place of serial computers.

There seems to be a contradiction here, but perhaps I am misreading the situation.

All the work I have read on eigenvalue techniques for parallel algorithms is devoted to treating full matrices. Yet most large matrices are sparse and one would expect any efficient algorithm to exploit such structure, if only to conserve storage

Very small changes in the assumptions concerning a large problem can strongly affect the efficiency of rival techniques.

Here is one difficulty. How can systolic algorithms with a very uniform flow of information respond efficiently to small changes in the specified task or the information available? If they cannot respond flexibly, then can they be effective for big calculations?

IMPACT OF VLSI ON MODERN SIGNAL PROCESSING*

Sun-Yuan Kung
Department of Electrical Engineering - Systems
University Park
Los Angeles, California 90089

EXTENDED ABSTRACT

Driven by steep increase in the complexity of computations, processing speed requirements and the volume of data handled in various signal processing applications, modern signal processing research and development is undergoing a major revolution. On the other hand, the availability of low cost, high density, fast-speed VLSI devices has timely opened a new avenue for implementing these increasingly sophisticated algorithms and systems. Therefore, the future trend of VLSI signal processing research will require very close interactions between different disciplines in VLSI, computer engineering, and signal processing. This paper addresses three fundamental issues:

(1) impact of VLSI on signal processing methods,
(2) impact of VLSI on array processors for signal processing
(3) integrated design of signal processing systems.

The purpose is to provide a cross-disciplinary understanding of the modern signal processing techniques, parallel array processors, and VLSI technology potentials and constraints.

I. IMPACT OF VLSI ON SIGNAL PROCESSING METHODS

computation potential of VLSI will definitely have an important impact on modern signal processing methods.

High Performance Signal Processing Techniques

In terms of advanced signal processing research development,

*Research supported in part by the Office of Naval Research under contract No. N00014-81-K-0191; by the National Science Foundation under grant No. ECS-80-17081 and by the Defense Advanced Research Project Agency under contract No. MDA903-79-C-0680.

the processing techniques can be divided into two categories: the conventional transform based techniques and the modern (non-linear) spectrum analysis methods [1].

The invention of the FFT provided the first major impetus towards the advancement of signal processing. The technique has drastically reduced computing time for signal and image processing problems, and this important advantange has rapidly promoted the FFT based methods to become the dominant approach in current signal processing research and development [2].

From another perspective, FFT based methods, being batch-processing technique, can not claim the recursiveness and self-adaptivity offered by some other methods. For example, the Kalman filtering method has a convenient recursiveness property, and is very useful in many real-time processing problems. Moreover, for high resolution spectrum estimation problems the transform based methods in general can not yield a satisfactory resolution capability [1]. In contrast, with a little higher computation complexity, modern (non-linear) spectrum analysis methods often offer much better performance[3 - 6].

Massive parallelism

While a major improvement in device speed is foreseen, it is in no way comparable to the rate of increase of throughput rate required by modern real-time signal processing. In order to achieve such increases in throughput rate, the only effective solution appears to be highly concurrent processing. Since massive parallelism will have a major impact on the modern signal processing techniques, the traditional performance criterion will have to undergo a major modification. In several instances, VLSI has made several conventionally inefficient or impossible techniques desirable or even rather attractive. Conversely, several "fast" techniques have started losing ground under this technological impact.

In evaluating the degree of inherent parallelism those methods with frequency or spatial decoupling will often receive a greater advantage, as they are decomposible into independent subproblems. On the other hand, much of the work on parallel algorithms has historically concentrated on numerical linear algebra [7 - 9]. Consequently, the class of methods which are reducible to basic matrix operations are becoming preferred candidates [10], [11].

Parallel Algorithms

There has been a number of efficient parallel algorithms for solving linear equations, triangular and orthogonal matrix decomposition (i.e. for matrix inversion and least-square solution), eigenvalue and singular value decomposition, and linear recurrences. An immediate consequence of the parallel processing capability for matrix operations is that the modern high-resolution methods such as Maximum Entropy Method [12], Maximum Likelihood Method [13], eigenvalue decomposition methods [4 - 6] can be performed at higher speeds.

The preference on regularity and locality will have a major impact in deriving parallel algorithms, (cf. Section II.a). In our work, the two most critical issues - parallel computing algorithm and VLSI architectural constraint-are considered:

1. To structure the algorithm to achieve the maximum parallelism and, therefore, the maximum throughput-rate.
2. To cope with the communication constraint (cf. Sect. 2.a)so as to compromise least in processing through-put-rate.

Example: <u>A Highly Concurrent Toeplitz System Solver</u>[14]

In order to achieve full parallelism, we have to fully exploit the Toeplitz structure. For this purpose, we have proposed a new, pipelined version of the Levinson algorithm which allows the "reflection coefficients" to be computed in a pipelined fashion. This avoids the need of the inner product operations, in the original Levinson algorithm, and the total computing time is therefore reduced to Ø(N). The configuration of computing structure and its VLSI implementation in a joint project between USC and Hughes are discussed in [14], [15].

<u>Challenge of Modern Signal Processing</u>

From the viewpoint of advancing signal processing research the overwhelming popularity on FFT-based methods has somewhat hampered the due attention to other (more analytical) modern methods. The latter approach, in our opinion, will have a long-term impact to the advanced signal processing technology. Here is our reasoning:

(1) Modern signal processing methods place the emphasis on certain critical optimality criterion (such as resolution), which will in turn lead to the optimal modeling and algorithms, at the expense of somewhat higher computational complexities comparing to the conventional FFT methods [16]. Consequently, they often provide better performances.

(2) More promisingly, computations for the modern signal processing methods, are often reducible to basic matrix operations which have very high potential for VLSI parallel computing. Thus, there are several very bright aspects for the design of modern signal processing computing systems (discussed

in the next section).

(3) The advent of VLSI offers a new opportunity to review the new signal processing trend. The low-cost, large volume of computations will encourage sophisticated, and high-performance techniques, at the expense of more computations. The new trend should not only encourage more in-depth mathematical analyses on modern methods traditionally deemed unfeasible; but also promote more hardware development of the modern fast-speed, special-purpose, parallel processors. The availability and accessibility of the modern computing hardwares will definitely further enhance the popularity on the modern signal processing methods.

II. IMPACT OF VLSI ON ARRAY PORCESSORS FOR SIGNAL PROCESSING

In order to accomodate the ever increasing speeds of processing requirements and volumes of data handled in various modern signal processing applications, the low cost, high density, fast VLSI devices has offered a new avenue implementing the modern signal processing systems. A very critical concern is that the traditional design of parallel computers and languages is deemed unsuitable for VLSI systems [17], since it suffers from heavy supervisory overhead incurred by storage, communication, and scheduling tasks, which severely hamper the real-time signal processing speed. Summarized below are several key considerations unique for VLSI design environment.

Communication Constraints

Although VLSI provides the capability of implementing a large array of processors on one chip it imposes its own constraints on the system. Among them, the most critical is the restricted (localized) communication, since it costs the most in VLSI chips, in terms of area, time and energy [18]. In general, highly concurrent systems require this locality property in order to reduce interdependence and ensuing waiting delays that result from excessive communication. This locality constraint, discouraging the use of centralized control, leads to the utilization of distributed control and localized data flow on regular and modular array structures. The utilization of a repetitive modular structure also help reduce the high design and test costs in VLSI systems.

VLSI Array Processors

The above restrictions, imposed by VLSI, will render the general purpose array processor very inefficient. It is therefore beneficial to restrict the array processors to a special class of applications, i.e. recursive and local data dependent algorithms, to conform with the constraints imposed by VLSI. On the other hand, this restriction incurs little loss of generality, as a great majority of signal processing algorithms possess these properties.

One typical example is a class of matrix algorithms. It has recently been indicated that a major portion of the computational needs for signal processing and scientific computation problems can, in fact, be reduced to a basic set of matrix operations and other related algorithms. Therefore, a special purpose parallel machine for processing these typical computational algorithms will be cost-effective and attractive in VLSI system design.

For special purpose array processors, the key of designing computing structures is mapping algorithms into computing structures, while keeping a natural topological relationship between the mathematical algorithm and the computing structure. Typical such design examples of the VLSI oriented computing structures are the systolic array processor [19] [18, Chap. 8] and wavefront array processor [20]. They are becoming increasingly popular due to their simple and regular control/data flows and localized communications.

Example: Wavefront Array Processor[20]

The wavefront array processor (WAP) is conceived as a programmable variant of the systolic array, aimed at solving all the matrix algorithms. To create a smooth data movement in a localized communication network, we make use of the computational wavefront concept. A wavefront in the processing array will correspond to a mathematical recursion in the algorithm. Successive pipelining of the wavefronts will accomplish the computation of all recursions.

The wavefront concept provides a firm theoretical foundation for the design of highly parallel array processors and concurrent languages, and it appears to have some distinct advantages. With respect to the language aspect, the wavefront notion drastically reduces the complexity in the description of parallel algorithms. The mechanism provided for this description is the special purpose, wavefront-oriented language, i.e. the Matrix Data Flow Language (MDFL)[20]. Rather than requiring a program for each processor in the array, this language allows the programmer to address an entire front of processors. As to the architectural aspects, the wavefront notion leads to a wavefront-based architecture which preserves the Huyghen's principle, that ensures wavefronts never intersect. Therefore, a wavefront architecture provides asynchronous waiting capability, and consequently, can cope with timing uncertainties, such as local clocking, random delay in communications and fluctuations of computing-times. Therefore, the notion lends itself to a (asynchronous) data flow computing structure that conforms well with the constraints of VLSI. In a sense, the WAP is an optimal combination of the synchronous and dedicated systolic array, and the general-purpose (asynchronous) data-flow multiprocessors.

Parallel Languages

In order to meet the basic requirement of real-time processing rate and yet to accomodate a reasonable programmability, a new signal processing (parallel) language should be developed in coordination with the hardware structure.

Traditionally, programming language for multiprocessors focus exclusively on the description of parallel data executions with little consideration over the data movements. Therefore, programming an array processor usually involves a heavy burden of scheduling, resource sharing as well as the control of processor interactions. Due to the communication problem in VLSI systems, the issue of data availability and management becomes critical and it has become very desirable to have a new language capable of expressing parallel data movements in a computing network. In other words, an effective parallel programming language can not be simply an ensemble of separate programs; it should also precisely define the coordination and interdependence of the data and the sequencing of the tasks between the processing elements.

Moreover, a truly VLSI oriented parallel language should also take into account the features of regularity, locality, and data-flow nature in a VLSI array processor. For example, the design concept of the wavefront language MDFL [2] has largely complied with all the above considerations.

VLSI Signal Processor Architectures

For an optimal design signal the processor architecture there should be signal processing dedicated Control Unit, Program Memory, and ALU, etc. As to the ALU, the selections between fixed/floating point, parallel/serial, or cordic/noncordic arithmetic units depend largely on the applications. For example, Givens-type rotations have received a great deal of popularity in some new computing structures, such as lattice structures[14], least-square solvers[21 - 23] and eigenvalue-decomposition problems [23]. There are several candidates for the implementation of hardware "rotators", among them Cordic Unit [21], [24], [25] appears to be most competitive. However, in the rotations the major operations are square-roots and multiplications, for which the cordic may not offer as fast convergence speed as some conventional designs [26]. (Cordic is relatively more effective for computing trigonometric functions which are not necessarily useful in signal processing computations.) In short, the choice of signal processing arithmetic units (and in general signal processor architectures) remains largely unsettled, and its final settlement should take both the signal processing needs and VLSi implementations into account.

Fault Tolerance in VLSI Systems

While VLSI offers more affordable chip area, it is also becoming more vulnerable to faulty circuits. In order to achieve a more reliable VLSI computing system, fault tolerance techniques are considered. Basically, this is to trade chip area for reliability, yield, etc. From VLSI array processor perspectives, the regularity and locality properties should be exploited to design an efficient fault tolerant diagnosis scheme. Moreover, since repairing and rerouting are necessary, extra flexibilities on timing and data paths are often useful.

III. INTEGRATED DESIGN OF SIGNAL PROCESSING SYSTEMS

The future trend of technology will call for integrated research approach aiming at incorporating the vast VLSI computational capability into modern signal processing applications.

Top-down Design

Usually, a VLSI system design includes several design phases:

(i) Applicational Specifications, (ii) Theory and Algorithm, (iii) Computing Structure and Language, (iv) Processor Architecture, (v) Circuit Layout (CAD), and (vi) Fabrication/Testing.

In our opinion, a top-down design should start with a full understanding of the problem specifications, signal analyses, (parallel and optimal) algorithms; and then map them into a suitable computing structures. Once these steps are accomplished, several existing CAD packages may then be utilized to facilitate the circuit layout design to produce standard layout codes ready for chip fabrication. Finally, the tested chips should be integrated into the applicational systems. Here we emphasize a somewhat untraditional methodology calling for a close coordination of the algorithm, language, and hardware designs from The very beginning stages. It should be noted that there may be necessarily interactive (top-down and bottom-up) design activities across different levels of design phases, and very often several such iterations are unavoidable.

Development of VLSI Design Language

For hardware fabrication, register transfer or computer description languages have been developed as convenient models at levels of abstraction in digital design. These hardware description languages need to be extended for system level VLSI design. Naturally, VLSI systems should be described by multilevel, hierarchical models. By taking advantage of the modular and regular structure in VLSI systems and identify those modules functionally, rather than in terms of logic/circuitry details, high-level descriptions of VLSI systems should be possible [27]. One approach is to map an algorithmic level parallel language (such as MDFL) into its corresponding hardeare description codes. Such research effort, if successful, will certainly impact the future trend of customized designs of VLSI signal processing chips.

Insertion of VLSI Chips into System Design

One of the major difficulties in VLSI design is the integration of VLSI array processor chips into applicational systems. One of our current research objectives is to demonstrate the feasibility of utilizing the systolic or wavefront array processors to handle the major portion of parallel computations needed in some specific applications, such as real-time passive sonar systems.

IV CONCLUSION

The advent of VLSI technology provides an opportunity for setting a new signal processing trend. A cross-disciplinary and integrated (top-down) design methodology appear to be the most effective for the development of future VLSI signal processing systems. The paper presents some (rather subjective) viewpoints of the author's. This is indeed intended for inviting different or opposing perspectives, hopefully in a much broader spectrum, and yet focusing the theme on the VLSI signal processing research and development.

References

[1] H. Cox, "Resolving Power and Sensitivity to Mismatch of Optimum Array Processors", J. Acoustics, Soc. Amer., Vol. 54, No. 3, pp. 771-785, 1973.

[2] A.V. Oppenheim and R.W. Schafer, "Digital Signal Processing", Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1975.

[3] S.S. Haykin, ed. "Nonlinear Methods of Spectral Analysis", New York: Springer-Verlag, 1979.

[4] G. Bienvenu, "New Principle of Array processing in Underwater Passive Listening", These Proceedings.

[5] N.L. Owsley, "High Resolution Spectrum Analysis by Dominant Mode Enhancement", These Proceedings.

[6] R. Schmidt, "Mutliple Emitter Location and Signal Parameter

Estimation", in Proc. RADC Spectral Estimation Workshop, Rome, NY, pp. 243-258, 1979.

[7] A. Sameh, "Numerical Parallel Algorithm - A Survey", High Speed Computer and Organization, Academic Press, pp. 207-228, 1977.

[8] D. Heller, "A Survey of Parallel Algorithms in Numerical Linear Algebra", SIAM Review, Vol. 20, No. 4, pp. 740-777, Oct. 1978

[9] H.T. Kung, "The Structure of Parallel Algorithms", Advances in Computers, Vol. 169, pp. 70 - 111, Academic Press, 1980.

[10] S.Y. Kung, "VLSI Array Processor for Signal Processing", Conference on Advanced Research in Integrated Circuits, MIT, Cambridge, MA., Jan. 28-30, 1980.

[11] J.M. Speiser and H.J. Whitehouse, "Architectures for Real Time Matrix Operations", Proc., GOMAC, Nov., 1980.

[12] J.P. Burg, "Maximum Entropy Spectral Analysis", Ph.D. Dissertation, Stanford University, Stanford, California, 1975.

[13] J. Capon, "High-Resolution Frequency-Wavenumber Spectrum Analysis", Proc. IEEE, Vol. 57, pp. 1408 - 1418, Aug. 1969.

[14] S.Y.Kung and Y.H.Hu, "A Highly Concurrent Algorithm and Pipelined Architecture for Solving Toeplitz Systems", to appear in IEEE Trans on ASSP, 1982.

[15] J.G. Nash, S. Hansen, and G.R. Nudd, "VLSI Processor Array for Matrix Operations and Linear Systems Solution," Internal Report, Hughes Research Laboratories, Malibu, California 90270, March 1982.

[16] T. Kailath, "Modern Signal Processing", These Proceedings.

[17] L.S. Haynes, R.L. Lau, D.P. Siewiorek and D.W. Mizell, "A Survey of Highly Parallel Computing", IEEE Computer, Vol. 15, No. 1, pp. 9 - 26, January 1982.

[18] C. Mead and L. Conway, "Introduction to VLSI Systems", Addison Wesley, 1980.

[19] H.T. Kung, "Let's Design Algorithms for VLSI Systems", Proc. CALTECH Conf. on VLSI, pp. 70 - 90, Jan. 1979.

[20] S.Y. Kung, K.S. Arun, R.J. Gal-Ezer, D.V. Bhaskar Rao, "Wavefront Array Processor: Language, Architecture, and Applications", IEEE Transaction on Computers, Special Issue on Parallel and Distributed Processing, Vol. 31, No. 11, Nov. 1982.

[21] W. M. Gentleman and H.T. Kung, "Matrix Triangularization by Systolic Arrays", in SPIE Vol. 298, Real-Time Signal Processing IV, held at San Diego, California, Aug. 1981.

[22] H.M. Ahmed, "Signal Processing Algortihms and Architectures", Technical Report No. M735 - 21, Information System Laboratory, Stanford University.

[23] S.Y. Kung and R.J. Gal-Ezer, "Linear or Square Array for Eigenvalue and Singular Value Decompositions?", These Proceedings.

[24] J.S. Walther, "A Unified Algorithm for Elementary Functions", Spring Joint Computer Conference, 1971.

[25] H.M. Ahmed, "A Comparison of Arithmetic Units in VLSI Signal Processing Systems", These Proceedings.

[26] G. Sharma, R. Gal-Ezer, and S.Y. Kung, "A Note on Arithmetic Units for Signal Processing", to be submitted for publication.

[27] R. Travassos, "Development of a VLSI Design Language", Internal Report, Integrated Systems, Inc., Palo Alto, CA., 1982.

The Role of the CHiP Computer in Signal Processing*

Lawrence Snyder
Purdue University
West Lafayette, IN

ABSTRACT

The Configurable, Highly Parallel (CHiP) Computer is composed of a collection of parallel processing elements (PEs) placed at regular intervals in a two-dimensional lattice of programmable switches. Each PE is a microprocessor with its own local program and data memory and its own instruction counter; there is no shared primary memory. The switches, which have local memory, allow the processors to be dynamically connected into arbitrary topologies. The CHiP Computer can be thought of as a generic signal processor since it efficeintly hosts systolic arrays as well as most other recently developed signal processingalgorithms. The CHiP machine can be efficiently implemented in VLSI and is suitable for wafer scale integration.

*This work is part of the Blue CHiP Project. It is supported in part by the Office of Naval Research Contracts N00014 - 80 - K - 0816 and N00014 - 81 - K - 0360. The latter is Task SRO - 100.

A SYSTOLIC SIGNAL PROCESSOR FOR RECURSIVE FILTERING

Richard H. Travassos
Integrated Systems, Inc.
151 University Ave.
Palo Alto, CA 94301

ABSTRACT

This paper describes the design of a systolic signal processor for recursive Kalman filtering. The architecture of the processor is based on mapping the Kalman filter equations onto a linearly connected systolic array. A successful prototype of the systolic Kalman filter processor has been constructed using state-of-the-art VLSI components. The systolic processor is well-suited for recursive filtering, beam forming, target tracking, image processing and radar signal processing.

INTRODUCTION

Estimating the frequency and angle of arrival of a point emitter can be viewed as an application of recursive filtering. The signals received at the linear senosr array, shown in Figure 1, carry both angle-of-arrival and frequency information. The estimator in Figure 1 may be viewed as a space-time sampling system where the number of sensors, m, and the number of memory elements per sensor, n, determine the size of the two-dimensional window in the signal field. As in the one-dimensional case, the resolution that can be obtained with conventional Fourier analysis is limited by the window length, in both the temporal and the spatial dimensions.

In the one-dimensional case, Pisarenko has proposed a harmonic retrieval method for detecting sinusoids in additive white noise, that removes the bias problem associated with FFT methods while preserving high-resolution. Pisarenko's method, however, is not computationally efficient and must be applied off-line. Thompson derived an adaptive version of Pisarenko's method but its convergence rate is slow. Reddy and Kailath derived a least-squares type adaptive version of Pisarenko's method that is similar to the γ-LMS algorithm proposed by Frost for noise power cancellation of sinusoids in noise. A performance comparison of the adaptive Pisarenko method and a recursive maximum likelihood method indicates that the recursive maximum likelihood method gives higher resolution estimates but is computationally slow.

In this paper, systolic array concepts are used to develop efficient architectures to speed-up computations in the recursive maximum likelihood estimator. A systolic signal processor based on these concepts has been constructed and successfully applied to harmonic retrieval problems.

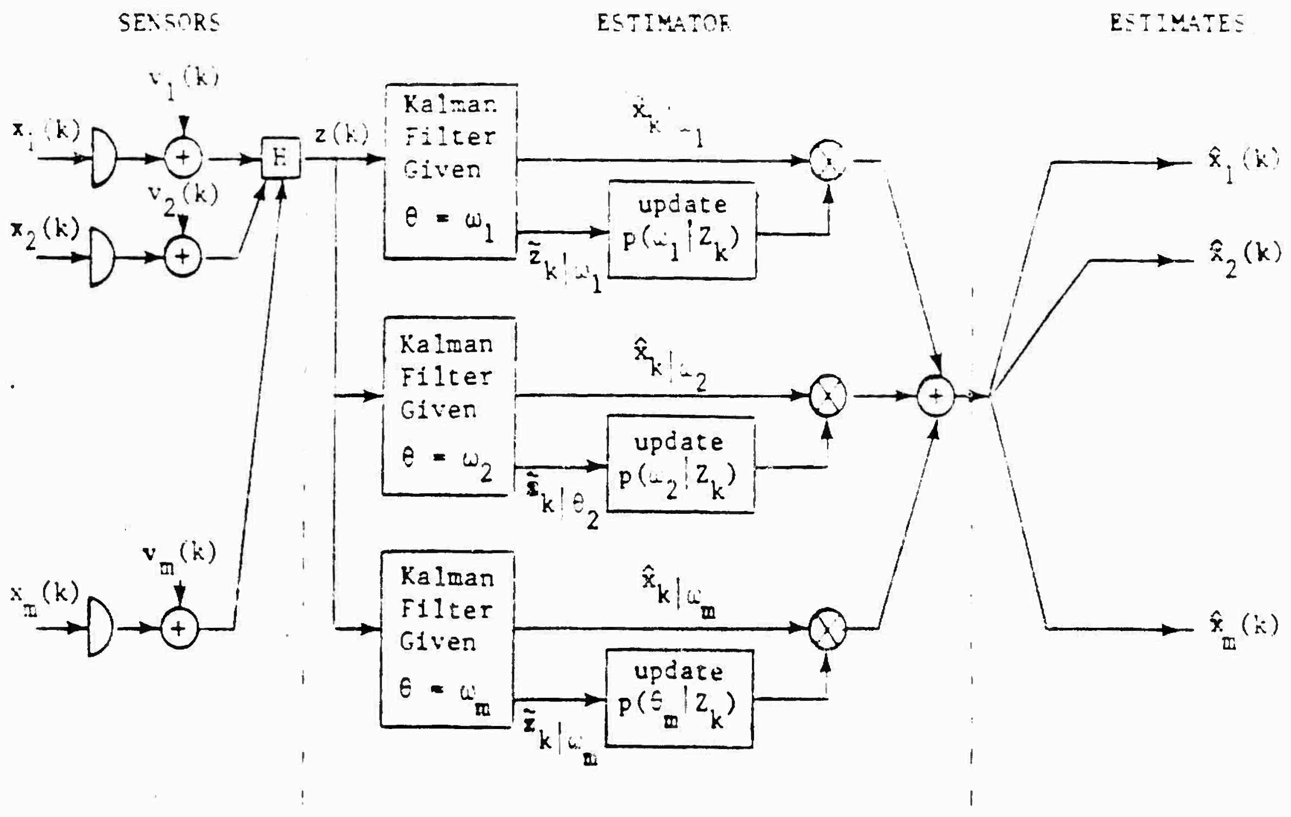


Figure 1. Maximum Likelihood Estimation of Sinusoidal Signals.

PROBLEM FORMULATION

Consider the problem of estimating the frequency and angle of arrival of multiple sinusoids corrupted by wide band noise. Let $x_i(t_k)$ denote the k-th time sample at the ith sensor in the linear array of Figure 1 and suppose the sinusoidal signals are modeled by:

$$x_i(t_k) = A_i \cos\left(\omega_i(t_k - i\nu_i) + \phi_i\right) \qquad (1)$$

where $\omega_i = 2\pi f_i$ and A_i are the frequency and amplitude of the ith sinusoid. The wave number ν_i of the ith sinusoid is defined as:

$$\omega_i \nu_i = \omega_i \frac{d}{c} \sin \theta_i \qquad (2)$$

where d is the spacing between the sensors, c is the propagation velocity of the plane wave and θ_i is the angle of arrival with respect to the broadside direction. The phase, ϕ_i, of the ith sinusoid is assumed to be uniformly distributed over $[0,2\pi]$. In the model, $t_k = kT$ denotes the kth sampling time and T is the sampling interval.

The recursive Kalman filter needed in Figure 1 is of the form:

$$\hat{x}_{k+1} = \Phi\hat{x}_k + K[y_k - H\hat{x}_k] \tag{3}$$

or equivalently,

$$\hat{x}_{k+1} = \underbrace{[\Phi-KH \quad K]}_{\text{filter design matrix}} \begin{bmatrix} \hat{x}_k \\ y_k \end{bmatrix} \tag{4}$$

In this formulation, the elements of the filter design matrix are constant. Φ and K, however, are dependent on the specific values of ω_i $i = 1, 2, \ldots, m$. y_k are measurements of the received signal. The structure of the filter indicates that it can be implemented as a single matrix-vector multiply. The elements of the filter design matrix can be stored in a ROM to cover the anticipated range of ω_i. A bank of recursive Kalman filters may then be used to obtain the maximum likelihood estimates of the sinusoidal signals from noisy data. (See Figure 1.)

The conditional probabilities, $p(\omega_i|Z_k)$, can be easily computed recursively from the following formula:

$$p(\omega_i|Z_k) = c \det (R^{-1})^{-\frac{1}{2}} \exp \{\tfrac{1}{2} \tilde{z}^T_{k|\omega_i} R^{-1} \tilde{z}^T_{k|\omega_i}\} \cdot p(\omega_i|Z_k) \tag{5}$$

where c is a normalizing constant, independent of ω_i, chosen to ensure that $\sum p(\omega_i|Z_k) = 1$. The innovations, $\tilde{z}_{k|\omega_i} = z_k - H\hat{x}_{k|\omega_i}$, and the covariance of the innovations, R, needed to evaluate $p(\omega_i|Z_k)$ can be obtained from the bank of recursive Kalman filters. The maximum likelihood estimates of the sinusoidal signals can then be obtained by computing (see Figure 1):

$$\hat{x}_k = \sum_{i=1}^{m} \hat{x}_{k|\omega_i} \, p(\omega_i|Z_k) \tag{6}$$

Note that once $p(\omega_i|Z_k)$ $i=1, 2, \ldots, m$ have been evaluated, the value of ω_i which maximizes $p(\omega_i|Z_k)$ can be easily determined by ranking $p(\omega_i|Z_k)$. Thus, the range of ω_i in the bank of Kalman filters can be reduced to obtain a more accurate estimate of ω_i. The bank of recursive Kalman filters, therefore, may be adapted to the data being processed.

Since $p(\omega_i|Z_k)$ needs to be evaluated rapidly, $p(\omega_i|Z_k)$ may be stored in a ROM in a table look up fashion. Thus, the key to implementing the overall scheme illustrated in Figure 1 is to develop a high-speed Kalman filter processor. Due to the structure of the filters, systolic architectures may be developed to speed up the computations.

SYSTOLIC KALMAN FILTER PROCESSOR ARCHITECTURE

The structure of the recursive Kalman filter algorithm described in the previous section suggests that it can be implemented on a linearly connected systolic array (see Figure 2). This architecture assumes the filter design matrix coefficients, f_{ij}, may be precomputed and stored in a ROM for a given range of [illegible]. This assumption is not restrictive since 32K ROM's already exist (e.g., Intel 27E5). The coefficient data can be downloaded to high-speed RAMs and passed to a systolic Kalman filter processor (see Figure 2). The state estimates in the systolic Kalman filter architecture can be computed as follows:

```
FOR  i = 1 to m
    $\hat{x}_i^{(1)} = 0$
    $\hat{x}_i^{(j+1)} = \hat{x}_i^{(j)} + f_{ij} s_j \quad j = 1, 2, \ldots, n+m$
    $\hat{x}_i = \hat{x}_i^{N+1}$
NEXT  i
```

Since the filter design matrix is $(n+m) \times n$, the above recurrence can be evaluated by pipelining the x_i and s_j through $n+m$ linearly connected processors.

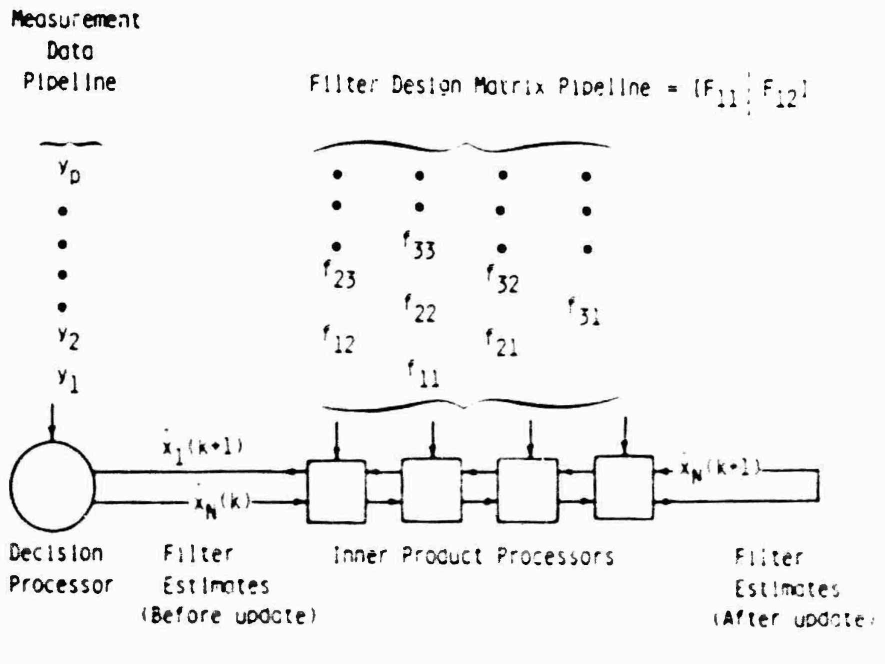


Figure 2. Systolic Kalman Filter Processor (n+m = 4)

BOARD-LEVEL IMPLEMENTATION

To determine the feasibility of the systolic architecture a board-level design was constructed. A 4K data memory was required to store the filter design matrix coefficients based on a 32 channel filter. On-board A/D and D/A converters were used to pass the signal measurements directly to/from the inner product step processors. Critical timing signals were generated by hardware, rather than by a microprocessor, to maintain a high-throughput rate. A 16-bit multiply-accumulator was used to compute the inner products in the architecture. A board-level prototype of one element of the systolic Kalman filter processor is shown in Figure 3. The prototype board has been operated successfullly at a 17.4 MHz clock rate. A faster clock rate should be achievable if the design is fabricated as a VLSI chip set.

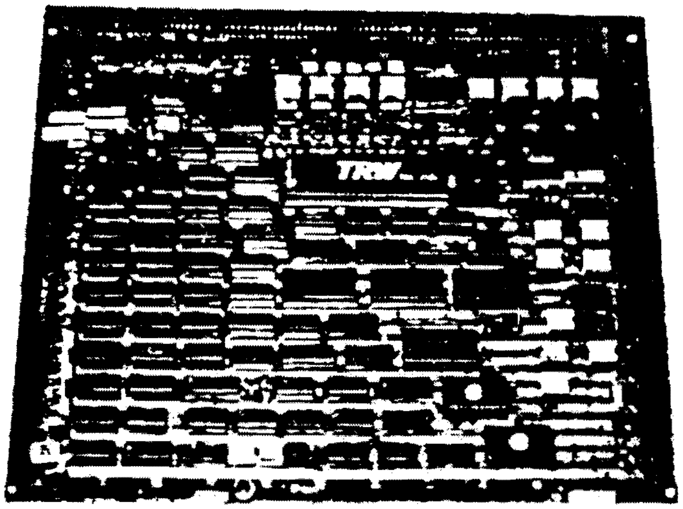


Figure 3. Systolic Kalman Filter Board.

SUMMARY

A systolic signal processor was developed based on mapping the Kalman filter recursions directly onto a linearly connected systolic array. An analysis of the recursive equations was used to specify the wordlength, sampling rate and memory requirements of the processor. The analysis showed that a 16-bit processor is adequate to ensure stability of the recursive equations for a wide range of applications.

A board-level implementation of the systolic signal processor was designed using state-of-the-art VLSI components. The systolic processor has been used successfully for harmonic retrieval and is currently being applied to other applications.

As VLSI technology matures, large numbers of processors may be fabricated on a single chip. At that time, it may be feasible to implement the board-level design as a VLSI chip set.

AN APPROACH TO RELIABLE PARALLEL DATA PROCESSING[*]

Gerard G.L. Meyer and Howard L. Weinert

Electrical Engineering and Computer Science Department

The Johns Hopkins University, Baltimore, MD 21218

*This work was supported by the Office of Naval Research, Statistics and Probability Program, Contract N00014-81-K-0813.

INTRODUCTION

Modern acquisition techniques produce high volume data sets that must be processed not only very quickly, but also reliably. The computational speed requirements can be achieved only through the use of either faster processing algorithms, or computational components with improved performance, or parallelism. The likelihood of further major advances in regard to the first two options is small. Therefore, the only realistic means for achieving high throughput is the use of parallelism.

Most signal processing algorithms are intended to be used on a single computational device, and therefore one needs specially designed segmented algorithms in order to use a network of interconnected computers. The segmentation of the software and the modularization of the hardware are not independent. In fact, software segmentation and hardware modularization must be considered simultaneously to achieve the desired efficient use of the hardware. The design of the interconnected network is a typical engineering problem in the sense that it involves a number of tradeoffs among speed-up, number of processors, number of interconnection links and maximum number of input/output links per processor.

In terms of reliability, we need the capability of efficient and timely fault detection, location, recovery and repair. Fault detection and location schemes should be capable of operating on permanent faults, transient faults, data dependent faults and environment dependent faults. As far as recovery is concerned, we want to immediately minimize the effect of the faults so that they do not propagate to the non-faulty hardware. A second constraint on the recovery schemes is that we lose as little data as possible. The repair schemes may use switching techniques to replace the faulty hardware, or to reconfigure the computing network so that the processing tasks may still be performed although in a possibly degraded fashion.

REAL TIME FAULT DETECTION

To achieve real time fault detection, one needs a concurrent fault detection scheme: one that takes place as the processing network is operating. There currently exist three basic techniques for carrying out concurrent fault detection. One is total hardware redundancy [AVI76], [GRA76], the second is partial hardware redundancy and the third is functional redundancy [SOG69], [CLA78]. In all three cases, the detection scheme involves the generation of an auxiliary boolean sequence $\{b(i)\}$, followed by a decision based on that sequence. The elements of $\{b(i)\}$ are generated so that if $b(i) = 1$, the system is faulty, and if $b(i) = 0$,the system may or may not be faulty. The decision scheme uses the elements of the sequence $\{b(i)\}$ up to the present time to determine the status of the system.

In the case of total hardware redundancy the entire signal processor P_1 is duplicated as P_2. At time i outputs $w_1(i)$ and $w_2(i)$ of P_1 and P_2, respectively, are compared to produce the quantity $b(i)$. The variable $b(i)$ is set equal to 0 if $\|w_1(i) - w_2(i)\| \leqslant \epsilon$, and is set equal to 1 otherwise. In the case of partial hardware redundancy, we do not duplicate the entire signal processor, but only part of it. This approach decreases the complexity of the hardware, but increases the design difficulty: the computational device P_2 is not identical to P_1 and must be separately designed. With functional redundancy, simulation is used in lieu of hardware duplication. The most important aspects of the operations of P_1 are simulated in the device P_2.

The above considerations lead to a general concurrent fault detection model in which, given a sequence $\{z(i)\}$ that is determined by the outputs of the computational elements of the signal processor, one needs to find a map $f(.)$ such that $f(z(i))$ is easily computed, $f(z(i)) \leqslant \epsilon$ means that the outputs may be correct and $f(z(i)) > \epsilon$ means that the outputs are incorrect. In general, there is no inherently natural map $f(.)$, and therefore we need to add hardware to artificially create such a map. This produces what we call an $f(.)$-redundant design. As long as no assumptions are made on the nature of the input data, one can obtain an $f(.)$-redundant design only by using total hardware redundancy, partial hardware redundancy, or functional redundancy. In the next section, we show that if the data is stochastic in nature, we can exploit this property to obtain a special type of redundant design, namely a stochastic $f(.)$-redundant design, that may be used for the purpose of fault detection.

STATISTICAL SIGNAL PROCESSING AND STOCHASTIC REDUNDANCY

For our purpose, we view statistical signal processing as the transformation of an input sequence of random variables $\{u(i)\}$ into an output sequence of random variables $\{v(i)\}$ possessing desirable properties. This transformation is implemented on a three part network: a generating network, a computing network and a receiving network. The generating network breaks up the input sequence $\{u(i)\}$ into N_g subsequences or input data streams $\{w_{0,0}(i)\}$, $\{w_{0,1}(i)\}$,..., $\{w_{0,N_g-1}(i)\}$. The computing network processes the input data streams in parallel and produces N_r output data streams. The receiving network combines the output data streams to produce the desired output sequence $\{v(i)\}$.

In order to achieve our fault detection objectives, the generating network should break up the input sequence $\{u(i)\}$ into subsequences having maximum mutual information [GEL59], [AND58], [BRI75]. Preliminary investigations have shown that under mild assumptions this goal may be achieved by the following natural scheme. For $k = 0,1,2,...,N_g-1$, let

$$w_{0,k}(jN_g+k) = u(jN_g+k), \; j = 0,1,2,\ldots \; .$$

The computing network is designed so that the initial redundancy (measured by the mutual information) among the input data streams is preserved as the data propagate through the network, and so that the necessary computations are distributed among the computing elements in a balanced fashion. In this way a stochastic $f(.)$-redundant design can be implemented without additional hardware, and the required processing speed can be achieved. A stochastic $f(.)$-redundant design is characterized by the following properties. If $w_{k,l}(i)$ and $w_{m,n}(i)$ are the respective outputs of any two computing elements $P_{k,l}$ and $P_{m,n}$, then there is an easily computed map $f(.)$ such that, in the absence of faults, $f(w_{k,l}(i),w_{m,n}(i))$ has zero mean and a variance that goes to 0 as i goes to infinity. As a result, the status of the computing network can be determined by comparing $f(w_{k,l}(i),w_{m,n}(i))$ to a threshold for various values of k, l, m, and n.

It is important to note that the need for real time fault detection puts nontrivial constraints on the way the data can be processed in parallel. Previous parallel processing studies [MCR74], [MOR79] have ignored the issue of network reliability.

REFERENCES

AND58 Anderson, T.W., *Introduction to Multivariate Statistical Analysis*, Wiley, New York, New York, 1958.

AVI76 Avizienis, A., Fault Tolerant Systems, *IEEE Trans. Computers,* Vol. C-25, No. 12, December 1976, pp. 1304-1312.

BRI75 Brillinger, D.R., *Time Series: Data Analysis and Theory,* Holt, Rinehart and Winston, New York, New York, 1975.

CLA78 Clark, R.N., Instrument Fault Detection, *IEEE Trans. Aerospace and Electronic Systems,* Vol. AES-14, No. 3, May 1978, pp. 456-465.

GEL59 Gelfand, I.M., and Yaglom, A.M., Calculation of the Amount of Information about a Random Function Contained in Another Such Function, *A.M.S. Transl.,* Ser. 2, Vol. 12, 1959, pp. 199-246.

GRA76 Gray, G.F., and Meyer, J.F., Algebraic Properties of Functions Affecting Optimum Fault Tolerant Realizations, *IEEE Trans. Computers,* Vol. C-25, No. 11, November 1976, pp. 1078-1088.

MCR74 Mc Reynolds, S.R., Parallel Filtering and Smoothing Algorithms, *IEEE Trans. Automatic Control,* Vol. AC-19, 1974, pp. 556-561.

MOR79 Morf, M., Dobbins, J.R., Friedlander, B. and Kailath, T., Square-root Algorithms for Parallel Processing in Optimal Estimation, *Automatica,* Vol. 15, 1979, pp. 299-306.

SOG69 Sogomonyan, E.S., Failure Diagnosis in Discrete Modular Objects, *Autom. Remote Control,* No. 10, October 1969, pp. 1688-1696.

Yield Enhancement by Fault Tolerant Systolic Arrays (Summary)

Robert H. Kuhn, Department of EECS

Northwestern University, Evanston, IL, (312)492-7193

Abstract

In this paper intersitial fault tolerance (IFT), a technique for incorporating fault tolerance into systolic arrays in a natural manner, is discussed. IFT can be used for reliable computation or for yield enhancement. Here ~~we~~ compare IFT used for yield enhancement to Wafer Scale Integration (WSI) techniques. Previous WSI techniques for yield enhancement have been proposed only for linear processing element arrays. IFT is effective for both linear and two dimensional arrays. Results of Monte Carlo yield simulation of IFT are presented.

1. Introduction

About the design of systolic arrrays for fault tolerance, we know very little. This is especially true in the area of yield enhancement techniques. Increases in the scale of integration result from two factors: decreasing minimum geometries and increasing chip size. One can delay fabrication of a device too large for current technology to take advantage of the former but the other alternative is very attractive because it can be attempted immediately providing an increased number of defects can be tolerated. Wafer Scale Integration (WSI) techniques [AbCa78, LeSr79, GaSa82, FuVa82] attempt the latter. Most approaches to WSI have been concerned with connecting the functioning units after fabrication. This results in a virtual one dimensional array which snakes its way around the wafer in an irregular but nearest neighbor interconnection avoiding defective units. (See Figure 3(a).) This technique is not suitable for two dimensional arrays, however. The interstitial fault tolerance technique permits the fabrication of one- or two-dimensional arrays in the presence of degraded yield in a straightforward manner.

In general, incorporating fault tolerance into a design implies redundancy. Redundancy may have two forms, increased hardware or increased computation time. Some WSI techniques [EWOT80], require increases in hardware and require overriding some of the efficiency of a fast local interconnection. The WSI techniques mentioned above maintain local interconnections but they implicitly use hardware redundancy in the form of unreachable functional units. The interstitial fault tolerance technique however uses time redundancy. Because implementing multiple processing element systems on a chip has only become feasible with VLSI technology and we seek to use this technology to implement highly parallel systems first and fault tolerant systems second few previous results hold.

Consider the conventional Price Yield Law [Pric70]

$$Y = (1 + \frac{DA}{m})^{-m}$$

where Y is the yield, D is the defect density, A is the effective chip area and m is the number of defect inducing mechanisms, mainly the number of fabrication steps. Price's yield law may not be realistic in this case for two reasons. First, it must be observed that processing elements are connected by only one or two layers of material in most systolic arrays so that the system can be modelled by a _two level model_:

- On the lower level, fabricating each unit or processing element does follow Price's law yielding a certain _unit defect density_, f, of nonfunctioning processors.

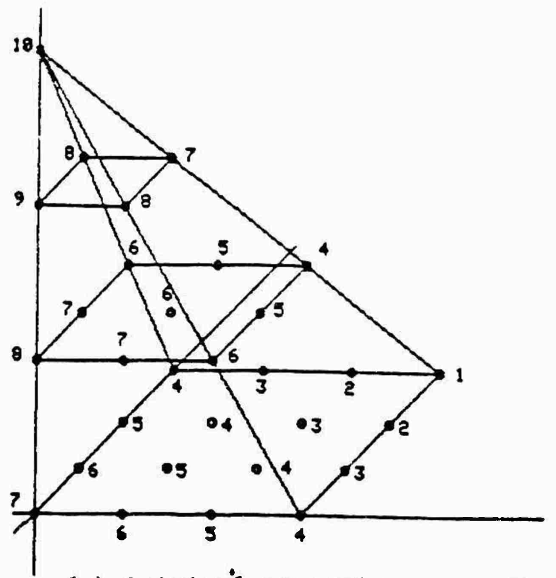
- On the higher level, the distribution of failed processors based on independent events obeying the unit defect density determines the _system yield_, Y.

Second, in conventional non-fault tolerant systems yield is actually the probability that zero defects occur in area A. For a parallel processor fault tolerant system such as the one proposed here this definition does not carry over. A large number of failures may be tolerated, up to O(N) out of N processing elements for interstitial fault tolerance, if the distribution is optimal. (In the WSI models mentioned above the yield degration is entirely graceful.) As determined by the two level model, the proper measure of yield for an interstitial fault tolerant system depends upon the local fault distribution. If the system will function with _k or fewer adjacent faults_, then the system yield of the fabrication process is the probability that the chip nowhere has more than k adjacent faults:

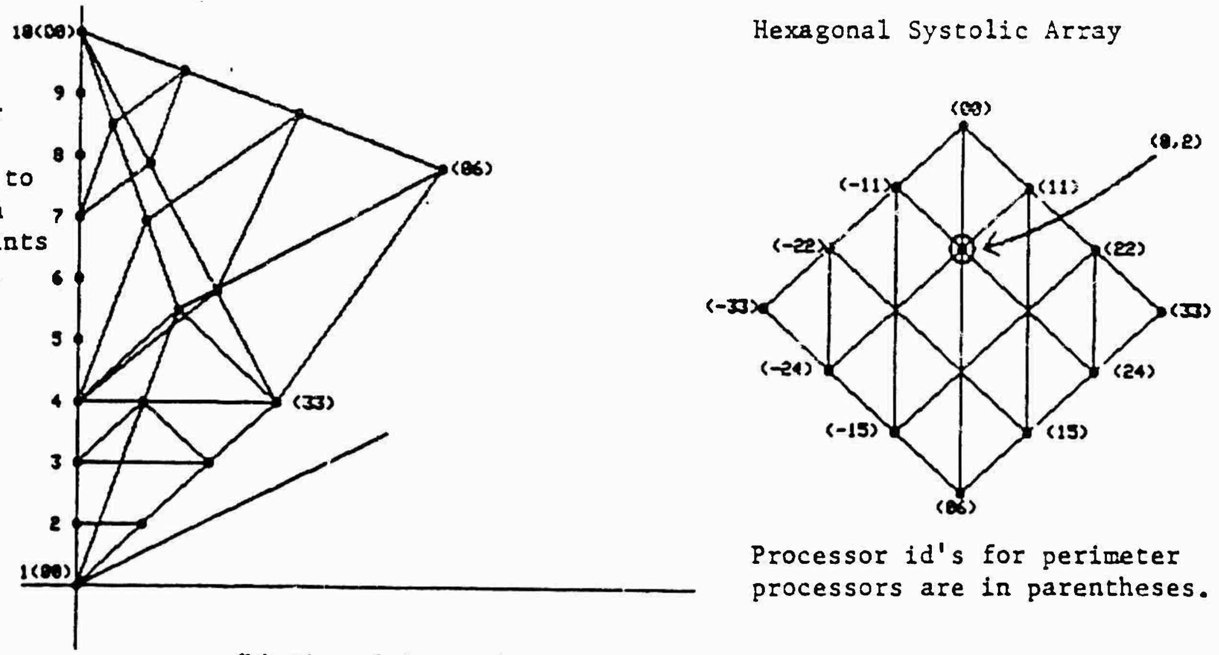
$$Y = Pr\text{ (defects occur in no more than k adjacent processors).}$$

2. _Interstitial Fault Tolerance_

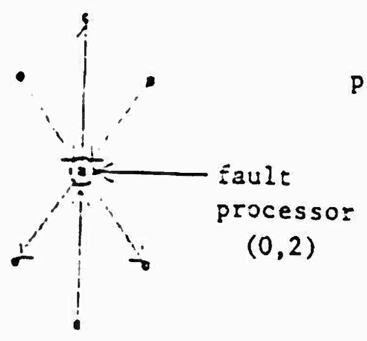
Any systolic algorithm can be represented by a skewed iteration space in which each operation can be assigned to a processor in space and a particular step in time. (A _skewing_ is a linear transformation from the iteration space to a processor-time space which exposes systolic parallelism.) Assume now that one processor in the array is faulty. As the systolic algorithm sweeps through the iteration space as a sequence of parallel wavefronts, one sequence or line of computations will not be performed. This line is called the _fault line_. It will be shown that by proper skewing of the iteration space the fault line can be mapped to another line of idle processor cycles which occurs on an adjacent processor. This line of idle cycles is



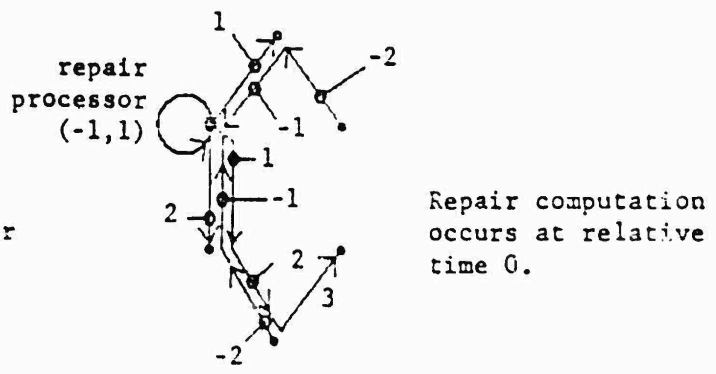
(a) Original iteration space for LU decomposition.



(b) Skewed iteration space.



(c) The local fault graph.



(d) Rippling the communications to avoid increasing link bandwidth.

Figure 1. Skewing the hexagonal LU decomposition algorithm for interstitial fault tolerance.

called the _repair line_, and the processor possessing the repair line is called the _repair processor_ for this fault.

In a properly skewed iteration space both fault and repair lines contain periodic compute and idle cycles. By design, at least one adjacent processor has idle cycles which mesh with the compute cycles of the faulty processor. Thus, the method is called _interstitial fault tolerance_. That is, the time redundancy needed for fault tolerance is hidden in the interstices of the prospective repair processor's computation. Interstitial fault tolerant arrays can be classified by a basic property of the skewing applied to the iteration space. If there exists n idle cycles per one compute cycle then the algorithm is called _level n inerstitial fault tolerant_. This figure is closely related to the system yield. A level n interstitial fault tolerant array will function with at least n adjacent faulty processing units.

Example: LU Decomposition

Figure 1 illustrates the application of this technique to Kung and Leiserson's LU decomposition algorithm. Figure 1(a) shows the original iteration space for the standard LU decomposition. Figure 1(b) shows how the iteration space is skewed to realize the algorithm on a hexogonal mesh. If processor (0,2) is faulty, the fault line of computations that would have been performed by processor (0,2) can be computed by processor (-1,1), the repair processor. The _repair line_ is an _interstitial repair line_. That is, no other computation is being performed at the times when a _fault line_ computation must occur. LU decomposition is level 2 interstitial fault tolerant. Figure 1(c) and 1(d) show the local fault graph and the local remapping of communication to effect the repair if the interconnection network at the faulty processor has failed. Three routing steps per computation cycle are required to mask the rerouting time in this case.

3. _Yield Analysis_

Yield analysis for interstitial fault tolerance has been conducted analytically and by simulation. Melzak [Melz73] and others have studied a mathematical model based on forbidden configurations and coincidences that can be adapted to evaluate the yield for interstitial fault tolerant arrays. A statement of the model in terms of our yield model is given by the parameters:

d-the systolic array dimension,

n-the number of faulty processors in the system,

f-a unit defect density function which holds for all defects, and

k-the maximum number of adjacent faulty processors.

Using the inclusion-exclusion principle, with this model it can be shown that:

$$Y_k = \sum_{r=1}^{\binom{n}{k}} \sum_{i=1}^{t(r,k)} (-1)^{r+1} N_{rik}(m) \int_{G_{ri}} \cdots \int f dx_1 \ldots dx_{v(G_{ri})}$$

where the G_{ri} are coincidence graphs with $v(G_{ri})$ vertices such as the ways in which triples of pairs of adjacent failed processors can share a failed processor, $t(r,k)$ is the number of such graphs for a fixed term r in the inclusion-exclusion series, and $N_{rik}(m)$ a combinatorial coefficient expressing the number of occurrences of the graph G_{ri} in the d times n dimension problem domain. However, evaluating this yield function is extremely cumbersome. Melzak points out that due to the oscillating and diminishing nature of the inclusion-exclusion formula the leading terms can be computed and an error bound proportional to the last term computed can be found.

Figure 2 shows the results of simulating three interstitial fault tolerant networks. The three networks simulated are:

o A linear systolic array with level 1 interstitial fault tolerance.

o A linear systolic array with level 2 interstitial fault tolerance.

o A linear systolic array with level 2 interstitial fault tolerance laidout in a snakelike manner on a mesh. In the linear snakelike network we allowed a processor in one row to repair a processor in an adjacent row. See Figure 3(b) for an example.

The method of simulation was to inject uniformly distributed random faults into the network. Each fault failed exactly one working processor. An adjacent processor which could execute the processing needed by the failed processor during its idle cycles was selected as the interstitial repair processor. The adjacent processors were checked in the same order for each failed processor. If no adjacent processor with interstitial idle periods could be found, the entire network was considered failed and the trial was terminated recording the number of faults it survived. No attempt was made to reassign repair processors in an optimal way. This model models a reliable computation model closely but it is more pesimisstic than a yield model should be in that multiple defects per processor should be allowed.

The results of the simulation for networks ranging from 16 to 256 processors are shown in Figure 2. Figure 2 shows the unit defect density that can be tolerated for a system yield of .5. (System yield as a function of unit defect density curves are contained in the report.) The interstitial fault tolerance technique compares favorably with the WSI techniques cited in the references. Especially for the snakelike linear layout. For example Figure 3 shows an 8 by 8 processor array with random unit defect with a density of 29% corresponding to a unit yield of 71%. Using a simple WSI algorithm, a usable yield of 50% is achieved by the connection shown in Figure 3(a). A level 1 interstitial fault tolerant network laidout in a snakelike manner with the repair processor assignment shown in Figure 3(b) would yield a functioning system using all 71% of the unit yield. The WSI approach has inherently used hardware redundancy while the interstitial fault tolerant approach has used time redundancy because the array will run at half the speed. (More sophisticated WSI algorithms have been developed such as [GaSa82] but they also have

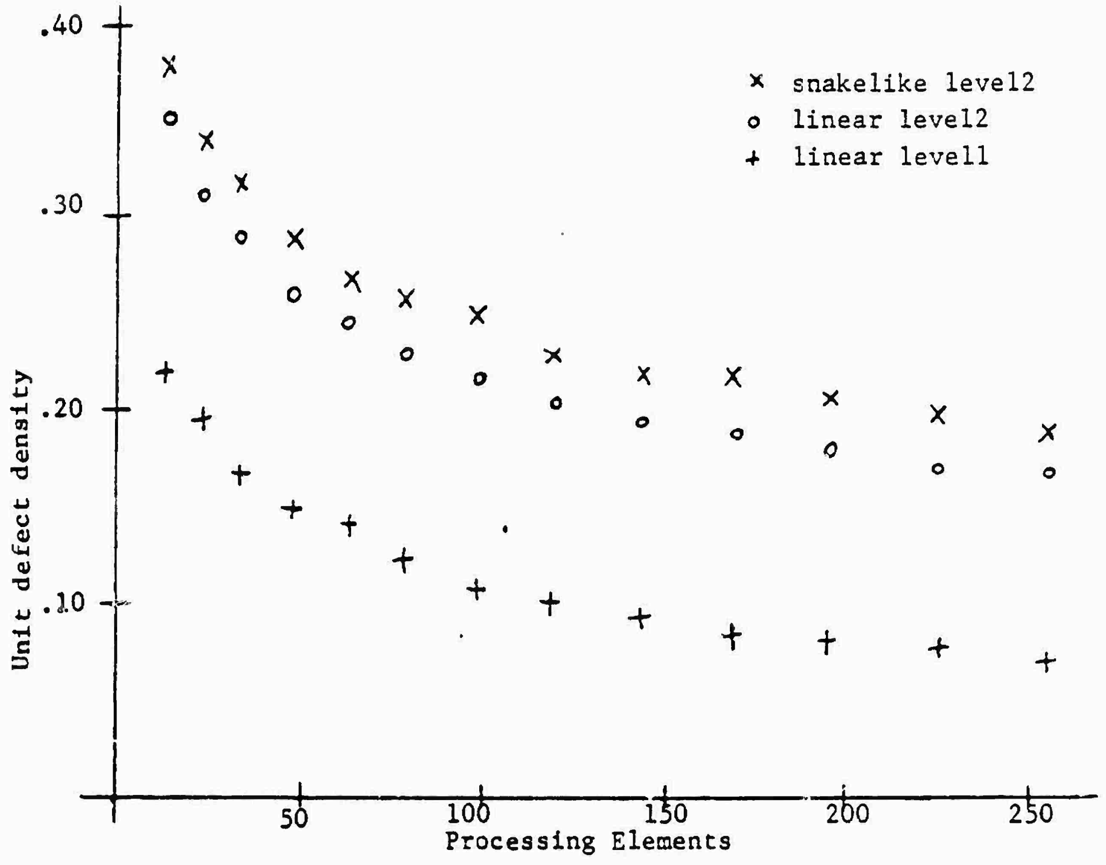


Figure 2: P.E. defect density vs. number of processing elements for system yield of .5.

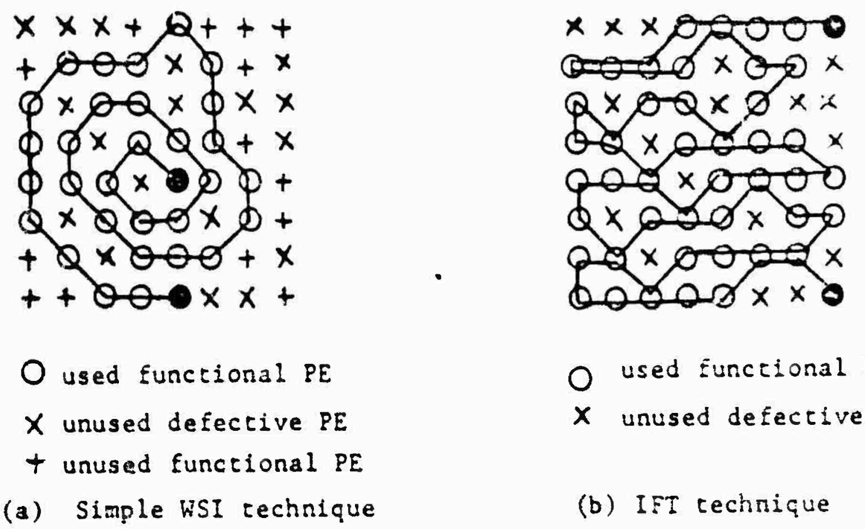


Figure 3: Comparison of WSI and IFT techniques applied to a random point defect distribution.

used some time redundancy.)

This discussion has centered around one type of defects, independent point defects, other types of defects have been observed on ICs such as: line defects, point cluster defects, and area defects. In addition, researchers have observed considerable radial variation in defect density [GuPl74]. That is, defects are more frequent on the perimeter of the wafer than at its center due to temperature gradients during fabrication, mask warping, and handling. Interstitial fault tolerance may be well suited to such distributions if the original iteration space is skewed so that the computation density decreases linearly towards the perimeter of the wafer. The full paper contains a two dimensional array that has been skewed to minimize the effect of radially increasing defect density The ability to adapt systolic arrays in this manner is due to our abstract representation of systolic arrays as skewed computations on an iteration space.

4. Conclusions

Highly parallel systems must provide some form of fault tolerance to be useful; especially in the VLSI area where defects are not uncommon. Interstitial fault tolerance is a natural technique for incorporating fault tolerance into systolic arrays and it compares well with currently known WSI techniques. The technique that we propose permits the linear speedup found in systolic arrays. And, using the analysis methods mentioned above, it can easily be used by an IC designer even after an initial layout of the systolic array without fault tolerance. Since interstitial fault tolerance uses time redundancy, it does not require large amounts of hardware dedundancy and it is especially well suited for configurable and programmable systolic arrays which already incorporate some form of local switches or routing tables. It also handles two dimensional arrays where WSI techniques do not.

References

[AbCa78] R. C. Aubusson and I. Catt, "Wafer Scale Integration - A Fault Tolerant Procedure", IEEE Journal of Solid-State Circuits, Vol. SC-13, No. 3, pp. 339-344, (June 1978).

[EWOT80] Y. Egawa, T. Wada, Y. Ohmori, N. Tsuda and K. Masuda, "A 1-Mbit Full-Wafer MOS RAM," IEEE Journal of Solid State Circuits Vol. CS-15, No. 4, pp. 677-686, (Aug. 1980).

[FuVa82] D. Fussell, P. Varman, "Fault-Tolerant Wafer-Scale Integration for VLSI," 9th Symposiumon Computer Architecture, pp. 190-198, (1982).

[GaSa82] D. Gajski, A. H. Saneh, "A WSI-Multiprocessor for Iterative Algorithms," Proceedings of GOMAC - 82, (Nov. 1982).

[GuPL74] A. Gupta, W. A. Porter and J. W. Lathrop, "Defect Analysis and Yield Degradation of Integration Circuits," IEEE Journal of Solid-State Circuits, Vol. SC-9, pp. 96-103, (June 1974).

[LeSr79] R. M. Lea and M. Sreetharan,
"VLSI Distributed Logic Memories," Proc. of Caltech Conference on Very Large Scale Integration, Caltech Computer Science Department, (Jan. 1979).

[Melz73] Z. A. Melzak,
Companion to Concrete Mathematics, Wiley and Sons, New York, N.Y., (1973).

[Pric70] J.E. Price,
"A New Look at Yield of Integrated Circuits," Proc. IEEE, Vol. 58, pp. 1290-1291, (Aug. 1970).

The Impact of VLSI on Signal Processing[1]

by

Danny Cohen
USC/ISI
Marina Del Rey, California 90291

Lennart Johnsson
Caltech
Pasadena, California 91125

Many modern applications, such as signal processing and real-time simulation, require large amounts of computation. This poses a dilemma: General-purpose computers are too costly for real-time applications, and special-purpose hardware is not flexible enough for research and development work.

Traditionally, Peripheral Array Processors (PAP) provide a cost-effective compromise that combines the best of both worlds: large amounts of computation performed at a very high rate, at a reasonable cost, and without loss of generality or programmability. Therefore, throughout this note PAPs are used as the baseline for the discussion of signal processing engines.

With the rapid advance of VLSI technology new alternatives have emerged, such as the so-called "signal processor on a chip" and numerous special purpose processing devices.

An interesting questions is, are the new VLSI devices about to do to the conventional signal processors what microprocessors and PAPs did to mainframes?

The Conventional Signal Processors

It is strange to use the term "conventional" to describe devices that have existed for only about a decade. In most other fields of technology 10 years is hardly a break-in period, but in electronic computation it is long enough for several generations to become obsolete.

Early Peripheral Array Processors were primarily signal processing engines. Since most signal processing algorithms are basically evaluations of bilinear forms, most of the PAPs were built around the concept of optimizing sum-of-products evaluated by a repetition of the multiply-and-add.

If fast hardware is to be used efficiently, it must be given data as fast as it can perform its operation. Hence, hierarchical memory caches and many parallel data paths are used in most PAP systems.

Generality results from the use of loadable microcode rather than hardwired instructions. Still, generality is below that of the instruction sets of most modern computers.

Additional I/O devices, more general instruction sets, large amounts of storage, and even peripheral devices have made recent PAPs more useful -- the well-known wheel of reincarnation is still turning. For example, early PAPs paid scant attention to such issues as address arithmetic, index registers

[1]This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under Contract No MDA903-81-C-0335, with USC/ISI. Views and conclusions contained in this note are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

pointers, indirect and other modes of addressing, subroutines, and logical instructions. They were limited in the amount of storage they could handle directly and had to rely on their host computers to provide interfaces to mass storage and to such external devices as sensors. In short, they were basically an augmentation of their hosts, aimed at fast arithmetic.

As experience increased, the importance of the above issues grew and became apparent. The transformation of PAPs recalls the evolution of general-purpose computers, which were once aimed primarily at numerical problems.

Many of today's PAPs are independent enough that they need very little help from a host machine. It is common to find a very small computer hosting a PAP with computational power orders-of-magnitude greater than itself. In such cases, the host primarily services man/machine interface, initial program loading, and similar low duty-cycle activities.

To sum it all up: PAPs started as small peripheral devices helping bigger hosts, but needing their help for I/O, mass storage and program storage. They quickly became big, independent and self-sufficient, even to having their own peripheral devices, which are more cost-effective in performing certain chores.

Progress in device technology has led to a new generation of general-purpose programmable mini peripheral array processors. These are much like the first PAPs, but at a much lower cost. Such a device typically consists of a card or two, designed to be installed directly not only on the buses of its host computer, but also physically inside it, hence sharing its power supply and cooling systems.

VLSI Processors

VLSI devices of two types are entering the array processing arena: special devices and general processors.

The former category include special-purpose chips designed for a specific function of the type that once required a full PAP, such as speech analysis and synthesis. modems and signal enhancement.

The latter category includes full-fledged general-purpose signal processors, typically optimized for general evaluation of bilinear forms. These can be programmed to achieve the functionality as the earlier PAPs, but not always with the performance of the existing PAPs.

Thanks to the low cost of VLSI devices large arrays of processors (either special- or general-purpose) can be implemented. Large arrays of even relatively slow processors may have high enough computational power for performing signal processing, as well as the conventional PAPs, which are typically single-engine signal-processors.

SPECIAL PURPOSE DEVICES

It is feasible now to implement, at a reasonable expense, VLSI devices optimized for solving particular problems. This is not so in situations in which the development cost of devices cannot be justified unless large number of devices are manufactured. However, since the new design tools and fabrication environment have reduced dramatically the cost of design and fabrication it is practical to implement VLSI devices even in small quantities.

In situation where performance and size are more important than flexibility special purpose VLSI devices prove to be the best solution.

Devices for real-time computer image generation and special filters are examples for such devices.

The well-publicized systolic arrays are also examples of special puroose high performance devices.

GENERAL PURPOSE DEVICES: VLSI/PAP

The VLSI/PAPs may not be as efficient as the special purpose devices, for some problems, but the flexibility provided by the ability to program the VLSI/PAPs often offsets the loss in performance.

Among the most interesting single chip programmable signal processors are TI's TMS-320, NEC's uPD7720, BTL's DSP, AMI's S2811 and Intel's 2920. Several others are in the final development stages, but not yet announced.

The designers of each of these chips made various trade-offs. For example, Intel put an analog subsystem (A/D and D/A conversion) on the chip, a choice not made by most of the others.

The designers of the NEC uPD7720 included memories for program, data (RAM) and such constants (ROM) as trigonometric tables needed for the Fourier transform. They also implemented a sophisticated and versatile I/O interface to support both parallel (under DMA control) and serial (up to 2.048 Mbps) communication with the outside. Not all designers chose to implement these features. TI, for example, excluded several of the above choices in favor of such features as a more general instruction set.

One cannot help noticing the steep learning curve demonstrated by these designers. They seem to have grasped nearly all the available lessons, and continue to assimilate new ones as they arise. These lessons are reflected in the new versions of VLSI/PAP chips.

As a result of increased attention to general system issues, most of the new VLSI/PAPs can be integrated into a multi-VLSI/PAP that delivers arithmetic instructions at a rate matched only by that of the few supercomputers and multiprocessors in existence.

ARRAYS OF PARALLEL PROCESSORS

Arrays of many independent processors working concurrently on the same problem present a possible dramatic increase in computational power.

Microprocessors (like RISC, the M68000 and the 8086) and VLSI/PAPs like the TMS-320 and the uPD7720) can be the building blocks of multiprocessor machines with hundreds or even thousands of independent parallel processors for applications such as image processing. Such multiprocessors would provide orders-of-magnitude more computational power than that available today, and provide it at a reasonable cost.

It is also possible to assemble arrays of special purpose nature in order to provide ultra fast solutions for specific problems. Many of the real-time multi-dimensional problems may, for the time being, be handled only by this approach.

However, it will be a long time before concepts in general and software in particular are developed to support these multiprocessor systems.

Cost-Effectiveness

Currently, most of the VLSI/PAPs are geared toward signal processing on relatively small data sets. This is not an inherent limitation of their architecture, but a result of particular implementation trade-offs.

There is no question that VLSI/PAPs represent a dramatic increase in cost effectiveness for signal processing applications. Still, they are far from reaching the maturity of first generation PAPs in ease of use, programming, simulation, and debugging. Though far behind in these areas, VLSI/PAPs are improving much faster than did their predecessors. However, the utility of the VLSI/PAPs is currently limited to those willing and able to expend the effort required to use them.

First generation PAPs perform fast computation much more cost–effectively than do mainframes, but this did not come easily. It took several years for PAPs to mature and come into wide use. It took a long time to develop the appropriate software (from microcode to the FORTRAN level programming capability), to learn to use PAPs effectively, and to educate users.

Mini PAPs are even more cost–effective than PAPs, but still are far behind on the learning curve. However, for many applications in which problems are relatively simple and well–defined, mini PAPs are very popular.

VLSI/PAPs are by far the most cost–effective of the above. They are perfect for special–purpose applications not requiring flexibility. These include components in consumer products and those applications in which programmability is only part of the design process (e.g., a modem) and is a given as far as the user is concerned. The consumer market, with its high volume demand, can reasonably be addressed only by VLSI technology.

In spite of the tremendous cost–effective advantage of the special-purpose VLSI device, their lack of programmability and the need to go through the entire implementation cycle (from design through fabrication and debugging to testing) for each case reduce dramatically, in our opinion, their expected impact on signal processing, except for a small number of very special application (mostly military).

We expect that because of issues like defered binding and quantities the same relation between general- and special–purpose devices which exists for computers will prevail also for VLSI devices for signal processing.

Future Trends

Forecasting is not as easy and safe as it used to be, but it seems reasonable to expect VLSI signal processing devices

- to be in the near future more of general purpose programmable nature (a la VLSI/PAP) than of special purpose nature;
- <u>not</u> to significantly affect the conventional signal processing arena until enough software is available for VLSI/PAPs, and many potential users are educated;
- to take some business away from the conventional signal processing engines, the PAPs but not to replace them entirely — just as PAPs did not entirely replace the mainframes.

In answer to our original question, Yes, the VLSI devices will do to conventional signal processors what microprocessors and PAPs did to mainframes.

◇

VLSI, SIGNAL PROCESSING, and FORMAL SEMANTICS

Carver A. Mead
California Institute of Technology

Signal processing has historically been the only widely practiced form of computing based upon a formal semantics. The properties of linear systems have allowed the behavior of a computational module to be defined independent of the input data upon which it will act. Although it is so familiar we all take it for granted, the linear transfer function is perhaps the most powerful engineering abstraction ever invented. Components can be composed by merely multiplying their transforms. It is commonplace for exceedingly complex filtering computations to be designed, implemented, and perform their function precisely as specified without a single redesign.

Contrast this situation with the common practice of computer programming. Specifications are vague. Most important aspects of an application are discovered in the process of implementation. Interfaces between modules are never precisely specified and cause enormous difficulty. Although advances in the art have been made in recent years through better modularity and more precise specification of interfaces, one astonishing fact remains – the only precise definition of the function of a system is the code which implements it! This situation is glorified with the title "Operational Semantics" in the computer sicence world, meaning "The Code Does Whatever It Does".

In the past when the world was viewed as one lone sequential process, it was perhaps possible to delude oneself into believing that a program was its own best definition. In today's world, algorithms are mapped onto silicon, distributed in space and time. The sequential way is no longer the only way – in most cases it is not even a viable contender. New approaches to algorithms are emerging daily, many of the best are represented at this conference. It is essential to be able to compare the area, time, and power required by several implementations of the same function. The ability to precisely specify the function performed by a system thus assumes far greater importance for the future than it has in the past.

In spite of the lack of formal specification and analysis, a small group of highly skilled professionals can build and maintain exceedingly ambitious software projects. It is instructive to observe how they do it. Universally, they take a hierarchical divide and conquor approach. The application is broken down into a small number of component modules. These are given names which clearly indicate their function. Each of these modules is further broken into sub-modules which are also carefully named. The process continues until the lowest level modues can be implemented directly. Complexity management is done through reasoning about a set of abstractions. The definition of each abstract function is carried by its name.

Since the earliest days of programming, attempts have been made to ascribe a precise definition to the abstract function performed by a program, apart from its implementation. If each part of a program performs an operation which is described by a mathematical function, the entire program can be viewed as the composition of its component functions. Powerful properties of mathematical functions can be brought to bear. This approach has evolved into the so-called applicative languages such as pure LISP, Bacus's FP, etc. A precise abstraction can be given for non-linear functions. However, one difficulty is encountered. There is no way to represent state. A bank account would be described by the entire history of deposits and withdrawals rather than by a simple balance. While functional notation provides an excellent mechanism for abstracting function, it does not allow for the abstraction of history! This inability to deal with state has prevented the purely functional or applicative approach from being widely used.

From a signal processing perspective, this state of affairs is incomprehensible. After all, a recursive filter retains traces of its input from an indefinite period into the past. The state stored in the delay elements of the filter is just the right abstraction of that history to give the filter its well defined transfer function. This point is not a trivial one – it clearly demonstrates that the presence of state does not, in itself, prevent a mathematical abstraction of the function.

We are led to ask whether we can, in some sense, have the best of both worlds – the power of general non-linear functions together with the linear systems' ability to abstract an element with state. The answer is yes, and details are given in Marina Chen's PhD thesis (CalTech, 1982). A brief account is presented for the first time in our companion paper in this conference. The success of this project is a direct result of its determination to understand and formalize real engineering practice used in the design of real systems. The key to understanding the dilemma came from signal processing. The Z-transform allows us to talk about events ordered in time. Since state is an abstraction of time history, no semantics can handle it properly if relations in time cannot be described. The purely functional approach ignores time completely, and thus cannot represent a system with state. In our treatment we combine the generality of the functional approach with the explicit time relationships necessary to represent state.

Armed with a formalism for describing and analysing general non-linear functions in a system with state, we are in a position to describe the broad range of real machinery which can be implemented in the VLSI medium. We can truly abstract from circuit and gate level to high level functions. I firmly believe that this approach will form the basis for a unified semantics of computation. However we must not underestimate the work left to be done.

Since the underlying space-time representation can describe any arbitrary system, it is in some sense like a blank piece of paper. It is as if we have just discovered differential equations, but have not as yet developed solution techniques. Of all possible classes of systems, only the familiar linear ones have evolved a sophistocated calculus. Using the space-time formalism for a wider range of algorithms is an important task for the future. I believe the effort will be well rewarded. For the first time we can discuss systems of all types, at all levels, in a common language.

AD P002622

VLSI ARCHITECTURES FOR RECOGNITION OF CONTEXT-FREE LANGUAGES[+]

by

K.S. Fu, Y.T. Chiang and K.H. Chu

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Summary

The speed of formal languages recognition is frequently considered to be important in many applications such as syntactic pattern recognition, artificial intelligence, natural language processing, syntax analysis of programming languages, pattern matching, etc. With the continuing advances in Very Large Scale Integration (VLSI) technology making circuitry smaller and faster, many processors can now be put together on a single chip and communicate with each other at on-chip speeds. This offers the opportunity in building low-cost, high-performance, special-purpose multiprocessor architectures to aid in the rapid solution of sophisticated language recognition problems. In this paper, two VLSI architectures are introduced for high speed recognition of general context-free languages. These languages are most commonly used in the mentioned areas and their recognition methods have been well studied. The recognition methods employed in this paper will be based on the Cocke-Younger-Kasami (CYK) algorithm and Earley's algorithm. Multiprocessing and pipelining techniques are used in the architectures to execute the algorithm in parallel.

The method proposed by Cocke, Younger and Kasami requires a grammar in Chomsky normal form with no null rule. The second major method was developed by Earley which will work for context-free grammar of any form. Although the two algorithms appear to be quite different, both have the same time bound of $O(n^3)$ for general context-free languages. Valiant had shown that the computation performed by CYK algorighm can be related to boolean matrix multiplication, and came up with a recognizer running in time $O(n^{2.81})$. This is the fastest known method of a sequential machine. About the same time, Kosaraju showed that CYK algorithm can be used to recognize context-free languages in time $O(n)$ on two-dimensional array automata, and in time $O(n^2)$ on one-dimensional array automata.

[+]This work was supported by the NSF Grant ECS 80-16580.

In order to implement the CYK algorithm efficiently in hardware, the VLSI structure is chosen to be the same as the strictly upper triangular recognition matrix T. In this way, the data paths which are determined by the recognition matrix and the algorithm are explicitly incorporated in the processors' organization. On the other hand, data in each matrix element are represented by using a s-bit bit vector as used in the preprocessing tasks. In this way, the set membership representation of nonterminals can be done efficiently. The hardware design can be subdivided into two portions: The dataflow requirement and the functional units design. The dataflow requirement takes care of the necessary data communications between cells whereas the functional units design handles the required operations on input data within a cell.

Each cell has three types of channel for data communications between its neighbors. The first one is called the fast belt, it can transmit one data value at a rate of one cell per time unit. The second one is known as the slow belt, it can transmit one data value at a rate of one cell in every two time units. The third one is a single bit control line, it is used for transmitting control signals between cells. There are five registers in each cell: The accumulator (ACCUM.) where the current value of of an entry of the recognition matrix is maintained, the horizontal fast (HORI. FAST) and vertical fast (VERT. FAST) registers for the implementation of the fast belts, and similarly the horizontal slow (HORI. SLOW) and vertical slow (VERT. SLOW) registers for slow belts implementation. Each of them is s-bits in length, where s is the number of distinct non-terminals in the grammar. In each time unit, the horizontal fast register receives data from its left neighbor while sending its old content to its right neighbor. On the other hand, the vertical fast register receives data from its neighbor below while sending its old content to its neighbor above. The horizontal slow and vertical slow registers behave exactly the same way except that the operation is done in two time units. That is, each of these registers has two stages. The incoming data enter the first stage, move to the next stage at the next time unit and finally exit the cell at the following time unit. In each unit of time, a cell takes part in the belt motion as well as updating its accumulator. The new value of the accumulator is updated by the functional module FM using the current contents of the five registers. The FM uses two permutation modules to compute all possible right-hand side nonterminal pairs induced by data in each register pair. In addition, if the cell is at distance t away from the boundary, then at time 2t it will copy the contents of its accumulator into its fast horizontal and vertical registers.

The VLSI system for Earley's algorithm is similar to the one for CYK algorithm except one extra vertical INP bus was used here to transfer input symbols. Every cell has identical structure. The system is controlled under a system clock (or unit time). It is assumed that during each system unit time, every cell can finish its computation and every bus can complete its data transfer operation. There are three vertical buses, VFB (vertical fast bus), VSB (vertical slow bus) and INP (input symbol bus), and two horizontal buses, HFB (horizontal fast bus) and HSB (horizontal slow bus). The fast buses and slow buses have the same transfer rate as discussed in the CYK algorithm, and INP bus has a transfer rate as that of a slow bus. Each cell has three functions, namely, computing the "x*" operation, loading the data onto fast buses and shifting the data from the fast buses to the slow buses. The last two functions are essential for keeping the bus system work and are controlled by two control lines, VC (vertical control) and HC (horizontal control). The control lines only transfer one bit at a time.

It is interesting to see that although the VLSI systems for CYK algorithm and Earley's algorithm use the same triangular matrix structure, the cell designs in each system are different. Notice that the VLSI system for CYK algorithm has simpler cell design; however, the algorithm requires the grammar to be converted into Chomsky normal form. On the other hand, a more complex cell design is required in the VLSI implementation for Earley's algorithm that, however, has imposed no specific restrictions on the form of grammar. For both algorithms, the designed VLSI systems are capable of recognizing a string of length n in 2n time units.

Since finite-state languages form a subset of context-free languages, the context-free parsing algorithms and hence the VLSI architectures described can also be used for the recognition of finite-state languages. A more efficient method for finite-state language recognition, derived from the CYK algorithm, has recently been proposed. The VLSI architecture using this algorithm can be made to recognize a string of length n in constant time.

EXTENDED SUMMARY

PARALLEL ALGORITHMS FOR IMAGE ANALYSIS

Azriel Rosenfeld

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

Image processing and analysis (IPA) systems employ a wide variety of techniques for image encoding, transformation, segmentation, and property measurement. Many of these techniques are suitable for efficient parallel implementation. This paper defines some general classes of IPA algorithms, and indicates how such algorithms can be implemented in parallel using various types of "cellular" multiprocessor architectures.

It was proposed about 25 years ago that many IPA algorithms could be implemented in parallel using a "cellular array" machine - i.e., a two-dimensional array of processors ("cells"), operating synchronously, each of which can communicate with its neighbors in the array. Several machines of this type, with array sizes of up to 128×128, have actually been constructed. Numerous IPA algorithms suitable for implementation on a cellular array have been developed. Recently there has been some interest in using "pyramids" of cellular arrays, of sizes $2^n \times 2^n$, $2^{n-1} \times 2^{n-1}$, ..., 2×2, 1×1, where each cell can communicate not only with its neighbors ("brothers") on its own level, but also with its four "sons" on the level below and with its "father" on the level above.

Cellular arrays or pyramids are suitable for many types of IPA operations at the pixel level. On the other hand, some image analysis operations, involving regions in an image, do not make use of pixel arrays, but rather use other types of data structures to represent regions and their relationships. For such operations, a more general class of graph-structured cellular machines would be appropriate, in which the cells correspond to the nodes of a graph, and can communicate with their neighbors as defined by the arcs of the graph.

For pixel-level operations taking images into images, a cellular array architecture, with processors connected in a regular grid, is very natural. For image property measurement, on the other hand, greater efficiency can be achieved by using tree-structured connections, with the processors at the leaves of the tree. For parallel processing of regions defined by border codes, ring-connected processors are appropriate. Other connection schemes are suitable if the regions are defined by maximal blocks, e.g., by run length codes or by quadtrees. At a more abstract level, when regions are represented by lists of properties, region merging can be carried out in parallel using a network of processors connected in the same way as the region adjacency graph.

Parallel region-level processing generally requires a much smaller number of processors than parallel processing at the pixel level. A cellular array machine for parallel processing of a 512×512-pixel image, one processor per pixel, requires $\frac{1}{4}$ million processors, which is not yet practical; but a region-level processor might require only a few hundred processors per region (depending on their complexity), or even fewer processors to handle a region adjacency graph (depending on the complexity of the segmentation). These numbers of processors are quite manageable, but their interconnections pose a problem. For pixel-level processing, the images to be processed will all be of the same size, and the neighbor interconnections are the same for every image, so that a cellular array machine can be hard-wired once and for all. For processing at the region level, on the other hand, the interconnections vary from image to image, since the shapes of the regions cannot be predicted in advance. Worse yet, we may even want the interconnections to vary in the course of a computation, as new regions are defined or old regions merged. This calls for some type of reconfigurable multiprocessor architecture, where ideally the reconfiguration itself should take place in parallel. For some types of representations (e.g., border codes, for which linked rings of processors can be used), such reconfiguration may be relatively easy; but for other representations, requiring tree or graph interconnections, parallel reconfiguration may not be easy to realize in such a way as to avoid serious interprocessor communication bottlenecks. As advances in hardware technology make it possible to build large multiprocessor networks, the problems involved in designing efficient systems for parallel image processing and analysis, both at the pixel and region levels, will have to be addressed.

Presented at the USC Workshop on VLSI and Modern Signal Processing, Los Angeles, California, November 1-3, 1982.

CONCURRENT VLSI ARCHITECTURES FOR TWO DIMENSIONAL SIGNAL PROCESSING SYSTEMS(*)

J.G. Nash and G.R. Nudd

Hughes Research Laboratories
3011 Malibu Canyon Road
Malibu, CA 90265

(*)Supported in part by NSF grant No. ECS 8016581 and Office of Naval Research contract No. N00014-81-K-0191.

Introduction

Because of the availability of low power, high density VLSI and recent developments in parallel algorithms, there presently exists a significant opportunity for dramatic advances in intelligent signal and image processing systems. The effect of these advances will be felt over a broad spectrum of commercial and military systems. Typical applications range from CAD/CAM for automation and robotics to intelligent communications, radar and image analysis systems.

[illegible] ~~We~~ concern ourselves here principally with the computational issues associated with detection and analysis of images, whether obtained by synthetic aperture radar (SAR), infrared radiometry, or visible imaging. We anticipate two types of developments. The first can be viewed as simple extensions of our present systems, such as enhanced resolution of radar and electro-optical imaging due to an increased processor throughputs. For example in SAR two basic filtering operations are required, pulse compression for range and doppler for cross range. One can show that the required throughput for range calculation is given approximately by 2BD, where B is the bandwidth and D is the duty cycle. The corresponding expression for cross range is

$$4\, v\, \Delta R \ln[R\, \lambda/2d^2]/d^2$$

where v is the effective platform speed, R is the range, λ is the transmitted wavelength, ΔR is the swath width and d the resolution. A future airborne mapping system, operating at W-band (94GHz), with one foot resolution over a mile swath could require throughputs $>10^3$MOPS for range data and >200MOPS for azimuthal data.

Once the two dimensional data has been correctly collected and formatted to provide conventional imagery, the subsequent processing steps are also highly computationally intensive. Typically, operations will include restoration and enhancement (potentially based on maximum entropy techniques or local matrix operations), followed by conventional feature extraction involving spatial filtering, median operations, histogramming, etc. Finally, some global operations, potentially including graph analysis, are required. Typically each element in the unprocessed image is processed in conjunction with its local neighbors over a kernel size ranging from 3x3 to 64x64. This involves each pixel being accessed and used up to 10^4 times. For relatively low resolution imagery (equivalent to television), with a raw data rate of 10^7 samples/sec, a throughput requirement of 10^5 MOPS is a minimum. Further, for operations such as median filtering, requiring $O(N^2)$ operations, or matrix based operations, requiring $O(N^3)$, throughputs of 10^7 MOPS or more are required.

A second, probably more important, system development will be the emergence of more intelligent and autonomous systems. A simple example might be a situation in which both the transmitter and sensor performances are made self-adaptive to optimize the overall system performance. This intelligence can frequently be gained by introducing multiple sensor systems working with loosely coupled nets of processors.[1] This concept of arrays of automated processors (each potentially working from a different sensor) with the capability of data 'fusion' is gaining popularity with military planners, and spawning new theories of detection probabilities.[2]

----

1. A. D. Shulman, "Correlation of Sensor Classifications: A State-Tracking Approach," IEEE Trans. Syst. Man. Cybern., SMC-8, July 1978, pp. 571-575.
2. P. J. Nahim & J. L. Pokoski, "NCTR Plus Sensor Fusion Equals IFFN or Can Two Plus Two Equal Five," IEEE Trans. Aerospace & Electronic Systems, AES-16, May 1980, pp. 320-337.

An essential ingredient for these future systems will be small, low-power processors with ultra-high throughput. The essential element to be exploited and the opportunity presented by VLSI is concurrency, both at the algorithmic level and in terms of hardware. Present indications are that a throughput improvement of 10^3 to 10^5 might be obtainable this way. Further, the pressing need to reduce memory access times and avoid complex data searching has been particularly emphasized in the need for a smooth and regular data flow.

Elemental Processor Design

Much attention has been given to various ways of arranging processor elements (PEs) in an array to simultaneously maximize their generality of capabilities and efficiency of processor usage. Examples of these are data flow, systolic, wavefront, and multiprocessors. However, less attention has been given to the problem of optimizing each PE for the most efficient use of area and time. The elemental processing unit must be very flexible, e.g., programmable, and capable of rapid performance of a few basic arithmetic algorithms. These two capabilities have in the past been incompatible. Generally one can either buy special purpose high speed hardware and integrate this into a processor, or use a microprocessor which contains sufficient bus oriented circuitry to be flexible, but is very slow. We are proposing a PE structure that integrates high speed arithmetic capabilities onto a slower system bus in such a way that both flexibity and speed are preserved. An example of this concept is shown in Figure 1. Here we have a dual bus running through our processor with various memory and sub-processor modules attached to it. The bus structure is the basis for the needed flexibility and the sub-processors provide the required high throughput. The sub-processors operate with their own set of high speed clocks in a serial-parallel fashion. The high speed clocks provide throughputs near those of parallel combinatorial logic, but with sufficient area efficiency to allow a large number of different modules on a single PE bus. Custom VLSI chips requiring arithmetic functions can be rapidly assembled by attaching required modules to the chip bus. The use of carry-save, bit slice schemes provide the flexibility necessary to increase operand lengths without incurring significant degradation in speed. Since the sub-processor modules run on their own separate clocks and are isolated from the bus in normal operation, it is possible for several of them to be operational at the same time, further increasing concurrency of operation.

Multiplication Oriented Processor (MOP) Chip

To demonstrate the integration of a high speed sub-processor module onto a bus, we have built a PE capable of high speed multiplications for application to signal processing systems where calculations of "ax+b" is of prime importance. The MOP chip we have designed contains 15,000 devices and was built for processor array applications. As shown in Figure 2, it is highly modular with a bus oriented, bit slice structure for ease of design and programmability. It uses a 28-bit fixed point word, and contains 10 dual port storage registers, a 32 word LIFO stack, a serial/parallel multiplier, a Manchester fast carry propagate adder, and two multiplexed I/O ports. The I/O ports allow parallel communication with adjacent chips at approximately 40M bytes/sec. The radix-4 multiplier, which runs on its own set of high speed clocks, uses a carry-save scheme to add partial products so that its speed is relatively independent of word length. All carry propagation takes place in the adder. The chip has been designed in a way that arithmetic and communication requirements between processor elements are "balanced" in the sense that neither causes a calculational bottleneck.

Applications

The MOP chip described above can be configured to perform a wide variety of concurrent cellular operations. An example which indicates the usage of this concurrency is the solution of Toeplitz linear systems.[3] Such systems are very common in both one and two-dimensional signal processing systems. For example in image analysis and restoration, the point spread function is of block Toeplitz form for space invariant systems. Toeplitz systems also appear in many in spectral analysis systems.[3] Included are autoregressive/maximum entropy analyses based on the autocorrelation matrix, and autoregressive methods operating directly on input sample data.

The Toeplitz system solver (TOPSS-28) which we are building solves, $\mathbf{R}\ \mathbf{X} = \mathbf{C}$, where X is an unknown vector, R is a known Toeplitz matrix, and C is a known vector. A modification of the Levinson algorithm is used to perform the LU decomposition of R.[4] This is done in a way that allows the reflection coefficients to be calculated without the evaluation of an inner product at each recursion. Thus, with O(N) processors, the Toeplitz system of O(N) can be solved in O(N) time steps instead of O(N ln N). The solution of $\mathbf{R}\ \mathbf{X} = \mathbf{C}$ can be shown to be $\mathbf{X} = \mathbf{U}^{-1}\ D\ [\mathbf{U}^{+}]^{-1}\ C$.[4] The matrix D is a diagonal matrix containing the diagonal elements of U. The multiplication by U^{-1} and $[\mathbf{U}^{+}]^{-1}$ is done in two back substitution steps, with the LIFO on the MOP chip used to store the elements of $[\mathbf{U}^{+}]^{-1}$ and return its transpose for the second back substitution

----

3. J. G. Nash, G. R. Nudd and S. Hansen, "Concurrent VLSI Architectures for Toeplitz Linear Systems Solutions," Proc. 1982 GOMAC Conf., Orlando Fla., Nov. 2-4.
4. Sun-Yuan Kung and Yu Hen Hu, "Fast and Parallel Algorithms for Solving Toeplitz Systems," in Proc. ISMM Int. Symposium on Mini and Microcomputers in Control and Meas., San Francisco, CA, pp.163-168, May 1981.

step. We use N/2 of our MOP chips (one processor per chip with 6um feature sizes) to solve an N^{th} order Toeplitz system. A pipelined organization with nearest neighbor communication and a global control arrangement are used. A special interface which we have developed is required for communication with the DEC unibus host computer.

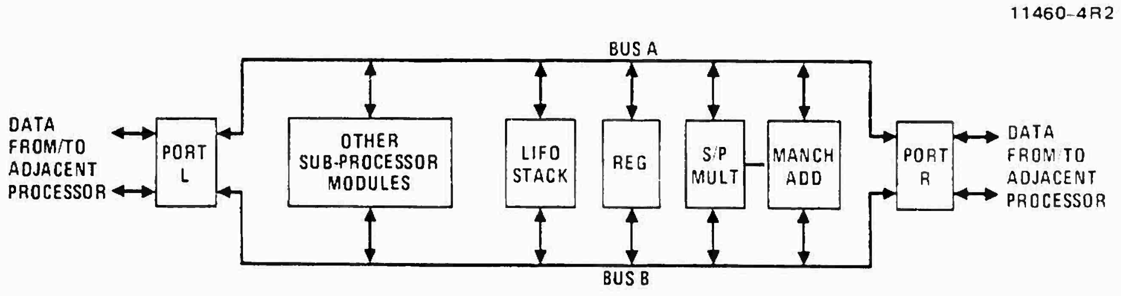

11460-4R2
BUS A
DATA FROM/TO ADJACENT PROCESSOR
PORT L
OTHER SUB-PROCESSOR MODULES
LIFO STACK
REG
S/P MULT
MANCH ADD
PORT R
DATA FROM/TO ADJACENT PROCESSOR
BUS B

Figure 1. Illustration of modular PE organization.

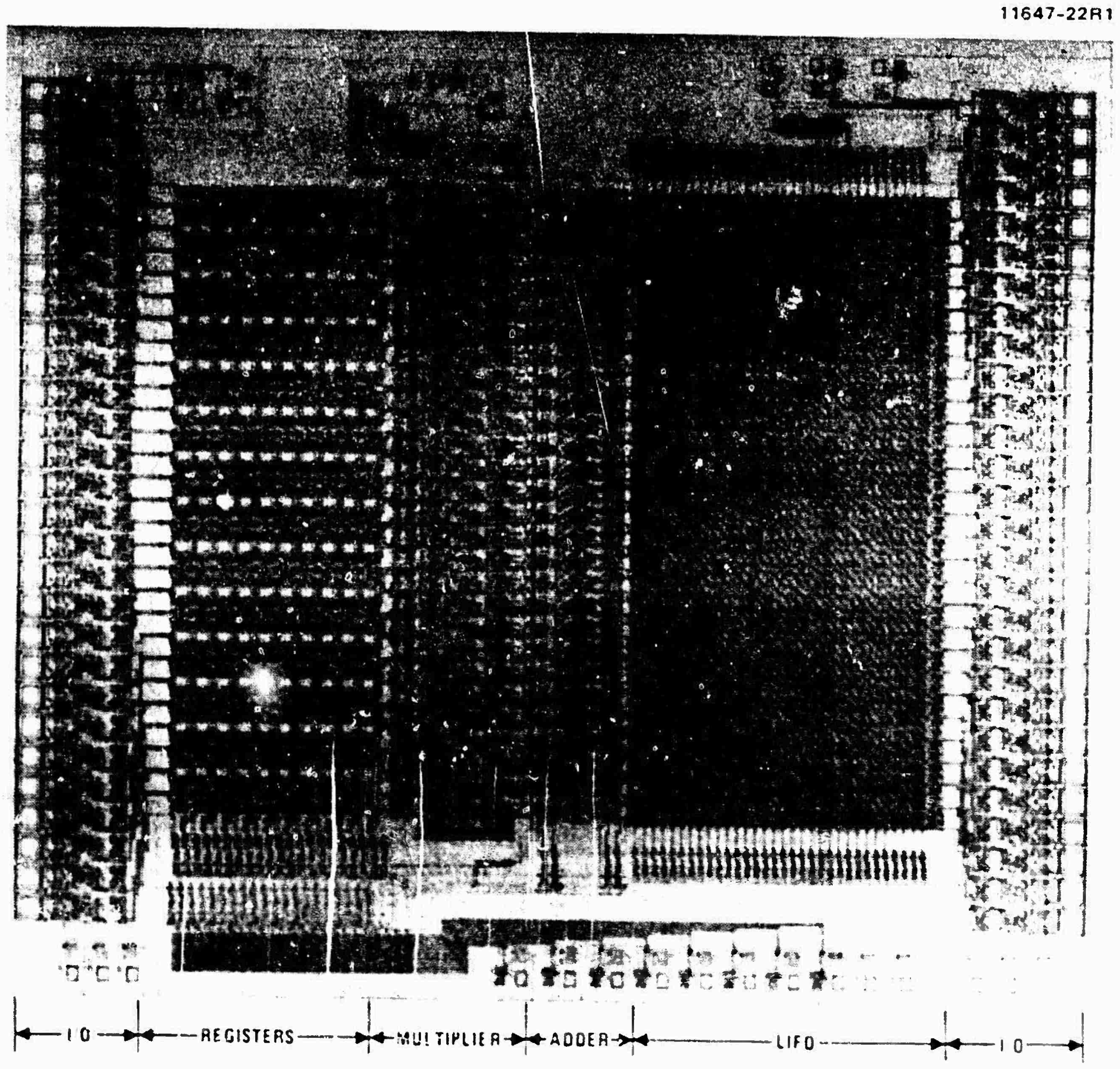


Figure 2. Layout of MOP chip.

VLSI Architectures for Pattern Analysis and Image Database Management

Kai Hwang
School of Electrical Engineering
Purdue University
W. Lafayette, Indiana 47907

In this presentation, VLSI computing structures are introduced for the analysis and management of imagery data. Information scientists have long recognized the fact that *one picture is worth a thousand words.* Machine intelligence would be greatly enhanced, if new generation of computers could be designed to process multi-dimensional imagery data above the string processing of alphanumerical information by present computers. Over the last decade, extensive research and development has challenged to achieve the capabilities of pattern analysis and image understanding by computers. Practical applications of such computers include the processing of biomedical images for diagnosis, the recognition of characters, fingerprints, and moving objects, remote sensing, industrial inspection, robotic vision, military intelligence, and data compression for communications [11,12].

Image analysis refers to the use of digital computers for *Pattern Recognition and Image Processing* (PRIP). On-line imagery data needs to be restored on disks and fastly retrieved for PRIP applications. A VLSI-based image analysis machine should integrate both pattern-analysis and image-database-management capabilities into a unified system design [1,10].

VLSI and Architectural Impacts

We are in the era of *Very Large Scale Integration* (VLSI). Integrated circuits have penetrated all areas of human civilization. It was reported that worldwide sales of electronic products reached 200 billion dollars in 1980, matching that of automobile sales at that year. IC wafer size has increased from 1 inch to 6 inches in twenty years. Bell Laboratory has produced some 8-inch wafers and 1 mega-bit memory chips. Fujitsu in Japan has announced a new non-silicon device, *High-Electron Mobility Transistor* (HEMT), which has a 17 pico-second switching time, 30 times faster than the fastest silicon counterparts. The *Very High-Speed Integrated Circuit* (VHSIC) project has challenged to produce 4-picosecond devices with some success. In the research community, *Wafer Scale Integration* (WSI) has been vigorously considered in implementing algorithmically specialized computer structures.

Advances in VLSI technology have triggered the thought of implementing many signal/image processing algorithms directly in specialized hardware chips. To promote image understanding, backend image database machine is highly desirable in future computers. The new concept of data-driven computations can be considered for artificial intelligence applications. New concepts on VLSI computing architectures and asynchronous dataflow multiprocessors should be seriously explored for potential use in image processing and pattern recognition. Extending control flow computers from 2-D arrays to 3-D pyramid structures is also a viable approach. Multiple-pipeline computers are also good candidate for parallel image processing, if task scheduling problems can be efficiently solved [3,6,7,8,10,13,14,15,16,18].

Image Analysis/Retrieval Functions

Deficiency of today's computers lies mainly in its input/output mechanisms. Still computers cannot communicate with human beings in natural forms, such as spoken or written languages, pictures or images, documents and illustrations. Existing computers are far from satisfactory in their slowness in I/O and lack of speech, vision, translation, and real-time responses. To develop "human-oriented" interactive computer requires first to upgrade their capability to understand "natural" information representations and to respond to them intelligently and perhaps more reliably than human beings. Natural language and speech processing are beyond the scope of this presentation. We focus below on developing intelligent computer with image analysis/retrieval functions.

To establish these functions, one need to develop subsystems for *input, output,* and *analysis* of imagery data retrieved from a large *image database system*. Pictorial functions of such an intelligent image processing computer are conceptually illustrated in *Fig. 1.* The input forms may be logic circuit diagrams, chest X-ray pictures, and aerophoto images. The corresponding outputs may be the layout of a VLSI circuit design, a precise description of the abnormality in the lung area, or a combat map generated in real time. The image database machine may be a part of a large knowledge base system. Many image analysis functions need to be built into the middle processing section for image enhancement and segmentation, feature extraction, pattern classification, structural analysis, image description and interpretation as listed in *Table 1.* Some of these functions can be implemented directly by VLSI hardware and some by special software packages. Advances in image I/O devices, pictorial query processing, and image database management techniques are demanded to construct an integrated image analysis/retrieval system [2,4,17].

VLSI Image Processors

Recently, many attempts have been made in developing VLSI devices for signal/image processing and pattern analysis. The statistical approaches to PRIP often involve large-scale matrix computations. The structural approaches require to perform syntax analysis and parsing operations. We present below as an example design of a pipelined VLSI architecture for statistical feature extraction. The pipeline stages are

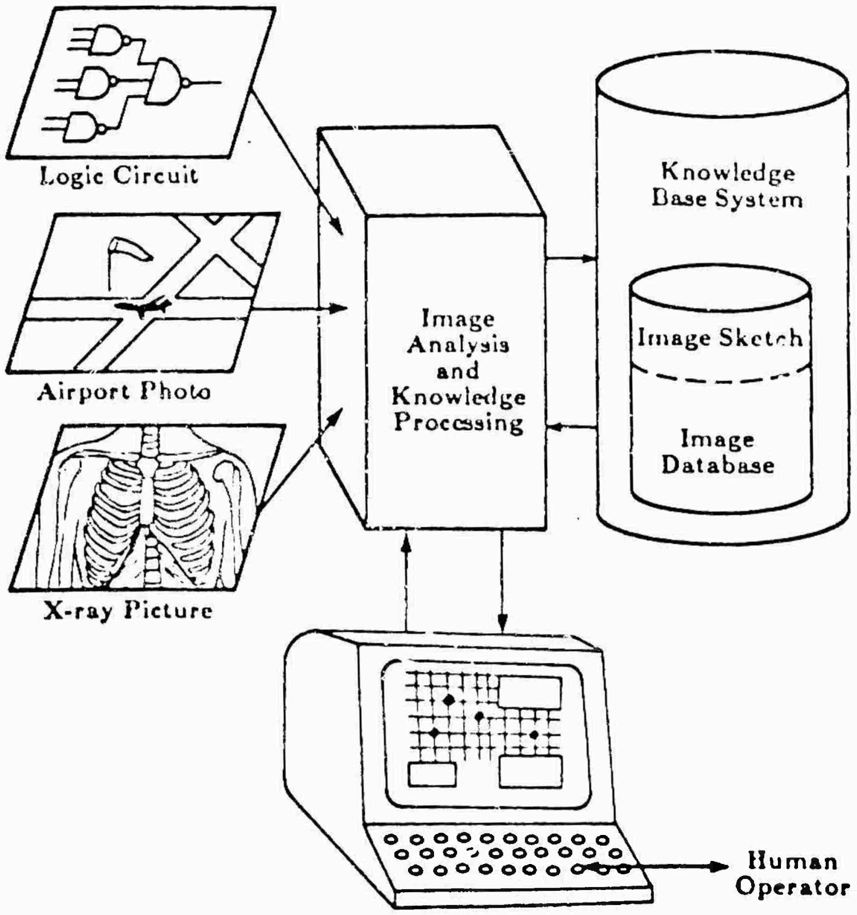


Fig. 1 Image Analysis/retrieval functions

Table 1 Candidate image algorithms for VLSI

Image Processing	Enhancement, Filtering, Fhining, Edge Detection, Segmentation, Registration, Restoration, Clustering, Texture Analysis, Convolution, Fourier Analysis, etc.
Pattern Recognition	Feature Extraction, Template Matching, Statistical Classification, Graph Algorithms, Syntax Analysis, Change Detection, Language Recognition, Scene Analysis and Synthesis, etc.
Image Query Processing	Query Decomposition, Query Optimization, Attribute Manipulation, Picture Reconstruction, Search/Sorting Algorithms. Query-by-Picture-Example Implementation, etc.
Image Database Processing	Relational Operators (JOIN, UNION, INTERSECTION, PROJECTION, COMPLEMENT), Image-Sketch-Relation Conversion, Similarity Retrieval, Data Structures, Priority Queues, Dynamic Programming, Spatial Operators, etc.

constructed with modular arithmetic devices as functionally specified in *Fig. 2.* These VLSI arithmetic modules will be used iteratively in submatrix computations. This pipeline architecture is based on the "partitioned" matrix algorithms developed in [9] for *L-U decomposition, matrix multiplication, inversion of triangular matrices*, and *solving triangular system of equations*.

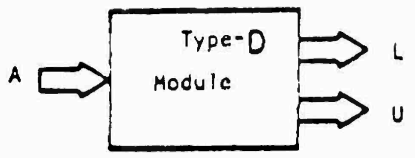
Figure 3 shows the pipelined structure for implementing the "partitioned" *matrix inversion* algorithm being outlined in four steps. Each VLSI module performs an rxr submatrix computation where n is the order of the input matrix U. In practice, n=kr and n>>r are assumed. The case of k=n/r=4 is shown in *Fig. 3.* For large k, this pipeline requires O(k) = O(n/r) VLSI modules to implement. The total time delay to generate $V=U^{-1}$ will be $O(n^2/r)$. Similar arithmetic pipelines can be constructed for *matrix multiply, L-U decomposition*, and *solving triangular systems*. Details can be found in [8].

Figure 4.a shows the functional design of a VLSI feature extractor. This extractor is constructed with three subsystems: ***scatter matrix generator, matrix inverter***, and ***feature generator*** as shown by dash-line boxes. The vector subtractor is implemented with modified V-modules for generating the sample offset matrices, and the mean difference. Two matrix multiply networks are used to perform orthogonal matrix multiplications. Each network contains n/r M-modules. The weighted matrix adder can be implemented by n/r M-modules with some special constant inputs. The inversion of the scatter matrix is done by employing an L-U decomposition network, two triangular matrix inverters, and one multiply network to yield the computation $\mathbf{A}^{-1} = (\mathbf{L} \cdot \mathbf{U})^{-1} = \mathbf{U}^{-1} \cdot \mathbf{L}^{-1}$. The feature generator can be implemented by V-modules with modified constant inputs. Finally, the matrix-vector multiplier is also implemented with V-modules.

Functional design of a VLSI pattern classifier is sketched in *Fig. 4.b.* The schematic design of the ***covariance matrix generator*** is similar to the scatter matrix generator in *Fig. 4.a.* The ***linear system solver*** is composed of an L-U decomposition network and a triangular system solver. This matrix solver is needed to triangularize a dense system. The Fisher classifier is implemented by some combinational logic circuits and modified V-modules.

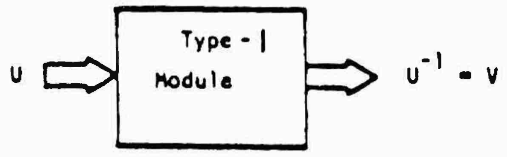
Image Database Machines

An image database system provides a large collection of structured imagery data (digitized pictures) for easy access by a large number of users. Most image database systems are implemented with specially developed software packages upon dedicated pattern analysis systems. It is highly desirable to develop a dedicated backend database machine for image database management. So far, several hardware attempts were suggested [1,16]. But none of them has been actually implemented for image database management.



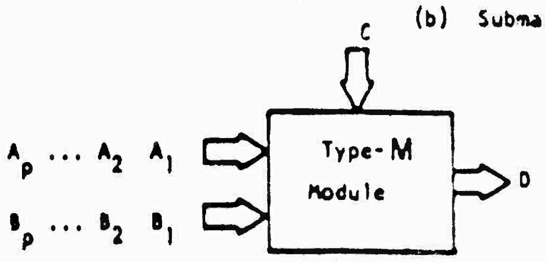
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

(a) Submatrix Decomposition Module



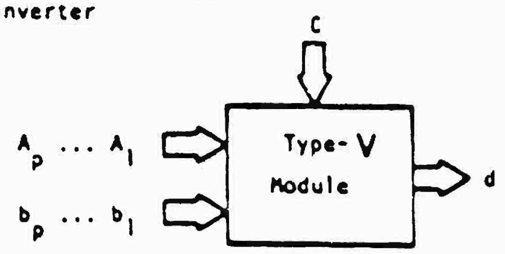
$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}^{-1} = \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ 0 & v_{22} & v_{23} \\ 0 & 0 & v_{33} \end{bmatrix}$$

(b) Submatrix Inverter



$D = C + \sum_{i=1}^{p} A_i \cdot B_i$ where C, D, and $(A_i$ and B_i for $i = 1,\ldots,p)$ are r x r matrices.

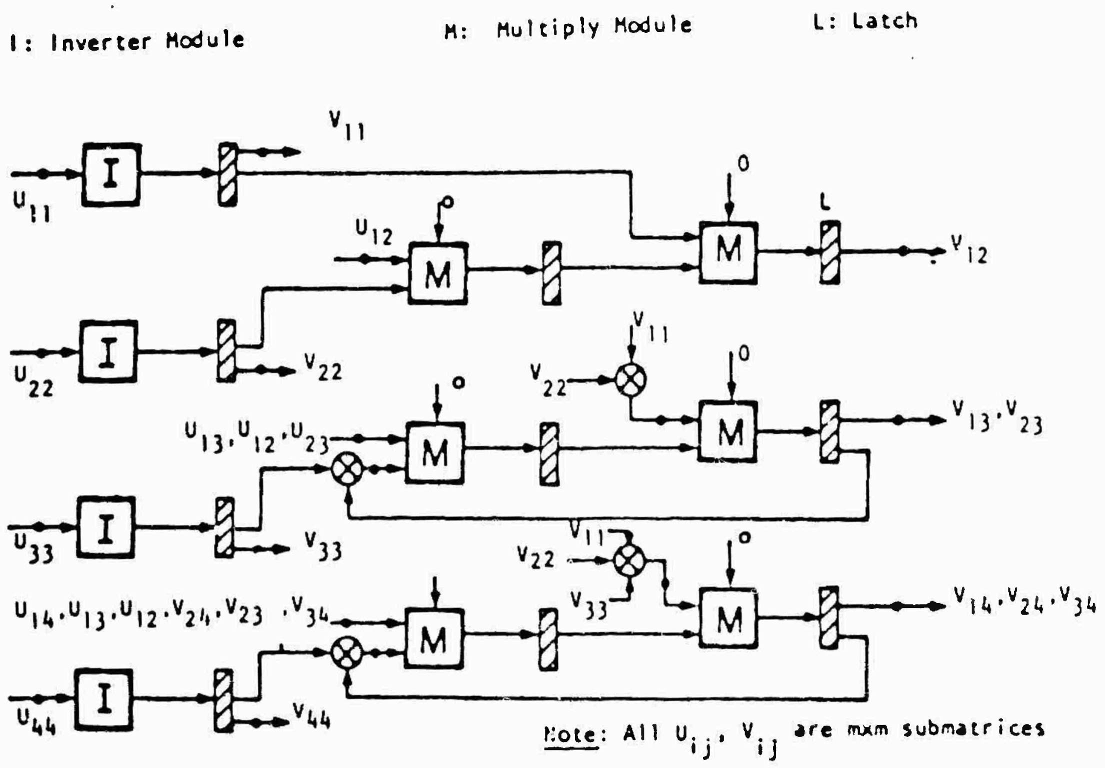
(c) Matrix Multiplier



$d = c + \sum_{i=1}^{p} A_i \cdot b_i$ where c, d, $(b_i$ for $i = 1,\ldots,p)$ are r x 1 column vectors, and $(A_i$ for $i = 1,\ldots,p)$ are r x r matrices.

(d) Matrix-Vector Multiplier

Fig. 2 Primitive VLSI matrix arithmetic modules



$$U^{-1} = \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}^{-1} = \begin{bmatrix} V_{11} & V_{12} & V_{13} & V_{14} \\ 0 & V_{22} & V_{23} & V_{24} \\ 0 & 0 & V_{33} & V_{34} \\ 0 & 0 & 0 & V_{44} \end{bmatrix} = V$$

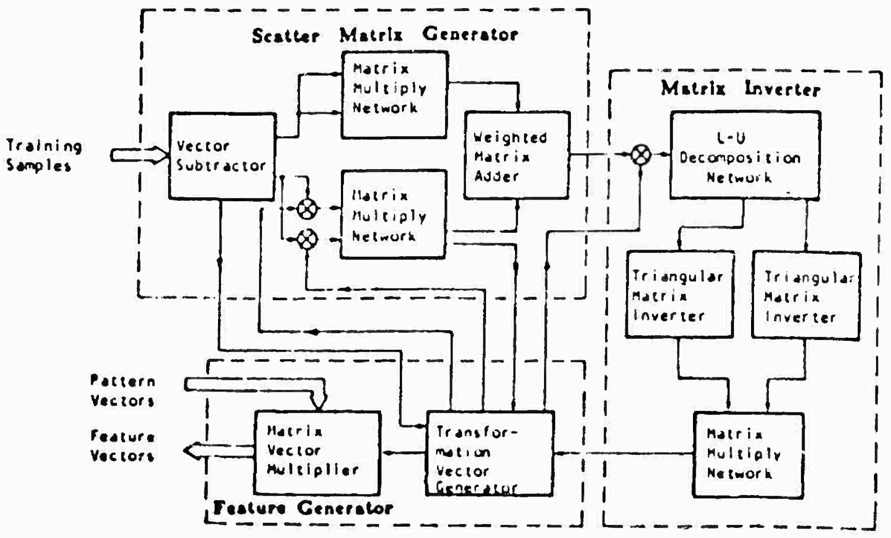
Step 1. $V_{11} = U_{11}^{-1}$; $V_{22} = U_{22}^{-1}$; $V_{33} = U_{33}^{-1}$; $V_{44} = U_{44}^{-1}$ (I-modules)

Step 2. $V_{12} = -V_{11} \cdot (U_{12} \cdot V_{22})$; $V_{23} = -V_{22} \cdot (U_{23} \cdot V_{33})$;
$V_{34} = -V_{33} \cdot (U_{34} \cdot V_{44})$ (M-modules)

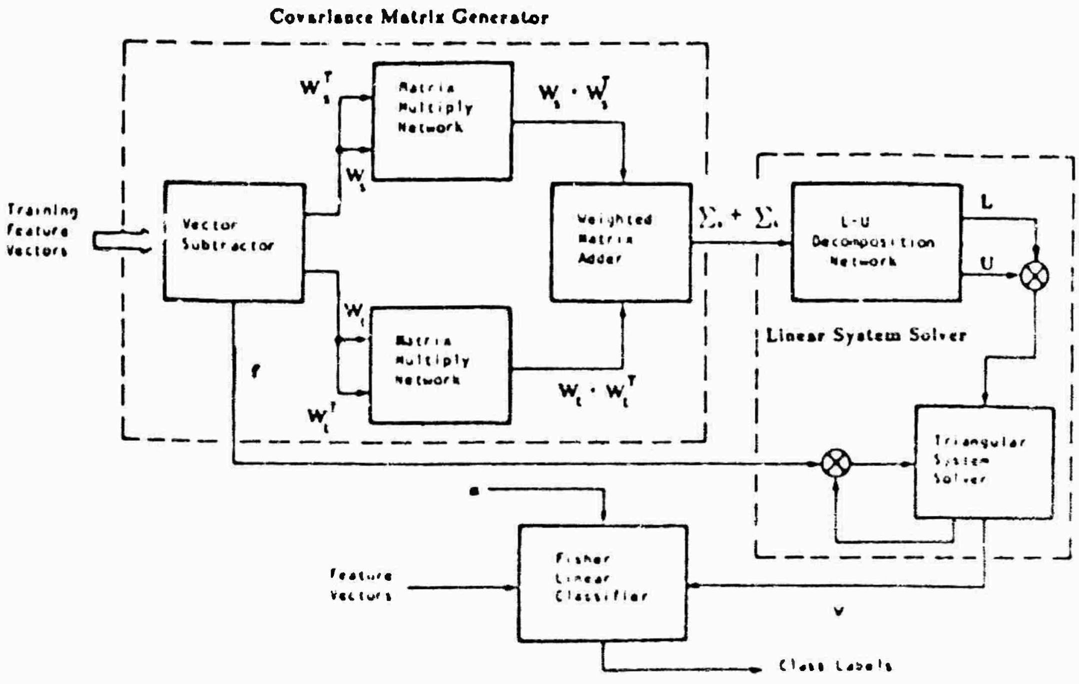
Step 3. $V_{13} = -V_{11} \cdot (U_{12} \cdot V_{23} + U_{13} \cdot V_{33})$
$V_{24} = -V_{22} \cdot (U_{23} \cdot V_{34} + U_{24} \cdot V_{44})$ (M-modules)

Step 4. $V_{14} = -V_{11} \cdot (U_{12} \cdot V_{24} + U_{13} \cdot V_{34} + U_{14} \cdot V_{44})$ (M-modules)

Fig. 3 VLSI matrix inversion pipeline based on Hwang/Cheng's partitioned algorithm.



(a) A VLSI feature extractor



(b) A VLSI pattern classifier

Fig. 4 VLSI architectures for pattern analysis/recognition

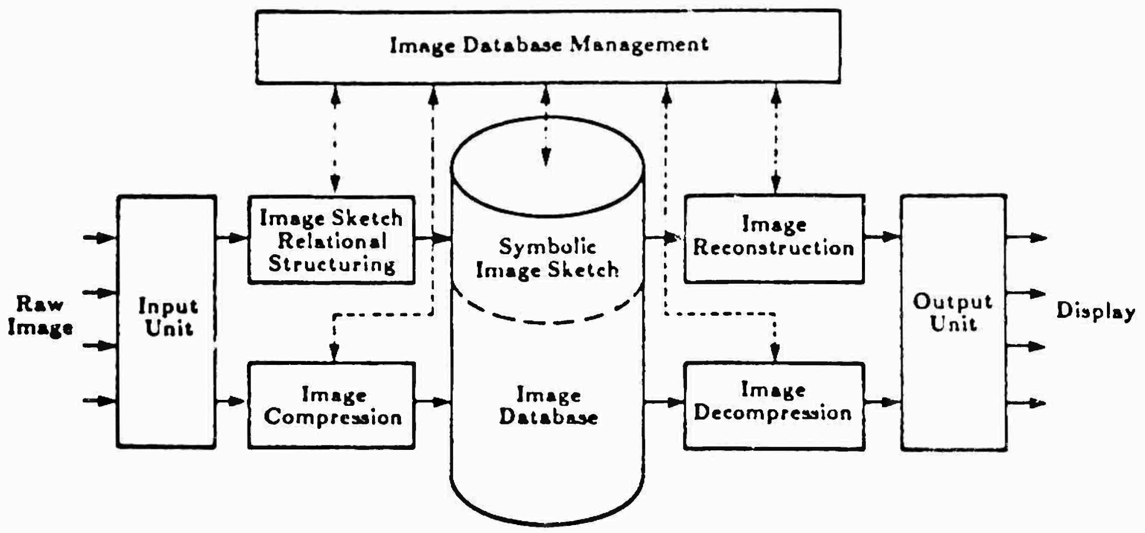
Image database management functions and peripheral supports are depicted in *Fig. 5.a*. First, we need faster and intelligent image input devices. The image features and structures (shape, texture, and spatial relationships) extracted by the host image processor should be converted into symbolic image sketches stored in a *logical* image database. For those unconverted raw images, the system must convert them into efficient codes stored in the *physical* image database. Flexible image manipulation and retrieval functions must be established using high-level image manipulation languages and image description languages. The logical database is used for image reconstruction from relational sketches. The compressed raw images must be decompressed for high-resolution console display. The above image database management functions should be implemented with specially designed image database machines at the backend.

Presented in *Fig. 5.b* is the conceptual design of a backend image database machine. It has a multiprocessor structure for parallel query processing and image database management. This structure is very similar to the DIRECT architecture [4], with multiple query processors accessing a set of shared data banks. A set of VLSI database functions is shared by all query processors. A resource sharing network is needed between the query processors and shared VLSI database operators. The shared database operators will be used for data filtering, projection, join or other operations, if relational image database is established. Some VLSI database operators are listed in *Table 1*.

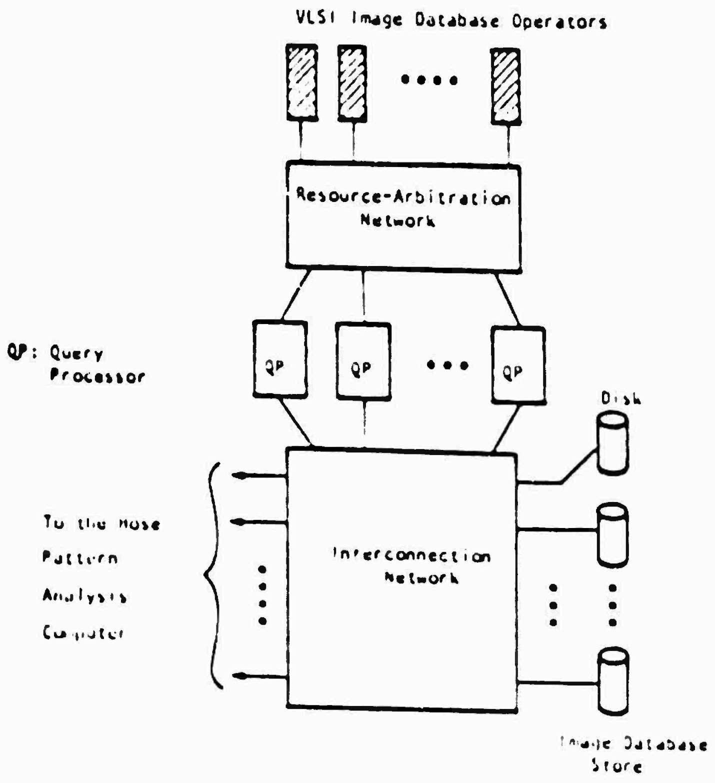
Concluding Remarks

Feature extraction and pattern classification are initial candidates for possible VLSI implementation. The Foley-Sammon feature extraction method [5] and the Fisher's linear classifier have been proposed for VLSI implementation [8]. Other methods such as the eigenvector approaches to feature selection and Bayes quadratic discriminant functions should be also realizable with VLSI hardware. It is highly desired to develop VLSI computing structures also for smoothing, image registration, edge detect , image segmentation, texture analysis, multi-stage feature selection, syntactic pattern recognition, pictorial query processing, and image database management, etc. The potential merit lies not only in speed gains, but also in reliability and cost-effectiveness.

Pattern-analysis and image database management are two functions that cannot be separated in an efficient pictorial information system. The integrated system approach is supported by the merging VLSI technology and by various parallel processing techniques. Cost-effectiveness is the key issue in developing special-purpose machines for image processing, recognition, and database management. Towards the eventual VLSI realization of an integrated image analysis/retrieval computer, we suggest below a number of important research topics:



(a) Image database management functions


VLSI Image Database Operators
Resource-Arbitration Network
QP: Query Processor
QP
QP
QP
Disk
To the Host Pattern Analysis Computer
Interconnection Network
Image Database Store

(b) A backend image database machine

Fig. 5 VLSI image database machine

(a) Develop systematic design methodology for mapping PRIP algorithms into VLSI hardware architectures.

(b) Develop VLSI devices for image description, image manipulation, and pictorial query processing.

(c) Develop backend image database machines, including both image database structures and management policies.

(d) Compression/decompression techniques for image data communication, storage, and display.

(e) Moving image processing, change detection, and scene analysis towards computer vision and related applications.

(f) Develop the resource arbitration networks for a multiprocessor system with shared VLSI resource pool.

(g) Promote biomedical image processing with specially designed systems. Remote sensing and earth resource satellite image processing applications.

(h) Innovative computer architectures for PRIP applications in artificial intelligence, industrial automation, CAD/CAM, and office automation.

(i) Reconfigurable architectural controls, partitionable interconnection network design, and macropipelining requirements.

(j) Effective resource allocation schemes for multiple pipelining, multiple-SIMD array processing, and asynchronous multiprocessing.

(k) Possible use of data flow concept in designing VLSI systems for PRIP and artificial intelligence computations.

(l) Integrate VLSI architectures for image analysis with those for natural language and speech processing.

(m) Extensions of 2-D arrays to 3-D structures for hierarchical image processing.

References

[1] Briggs, F. A., K. S. Fu, K. Hwang, and B. W. Wah, PUMPS Architecture for Pattern Analysis and Image Database Management, *IEEE Trans. Computers*, October 1982.

[2] Chang, S. K., J. Reuss, and B. H. McCormick, "Design Considerations of a Pictorial Database System," *Int'l. Journal of Policy Analysis and Information Systems*, Vol. 1, No. 2, Jan. 1978, pp. 49-70.

[3] Chu, K. H., and K. S. Fu, "VLSI Architectures for High-Speed Recognition of General Context-Free Languages and Finite-State Languages," *Proc. 9th Int'l. Symp. on Computer Arch.*, Austin, Texas, April 1982, pp. 43-49.

[4] DeWitt, T., "DIRECT: A Multiprocessor Database Machines," *IEEE Trans. Computers*, 1979, pp. 395-406

[5] Foley, D. H. and J. W. Sammon, "An Optimal Set of Discriminant Vectors," *IEEE Trans. Computer*, March 1975, pp. 281-289.

[6] Foster, M. J. and Kung, H. T., "The Design of Special-Purpose VLSI Chips," *Computer Magazine*, Jan. 1980, pp. 26-40.

[7] Hwang, K., and F. A. Briggs, *Computer Architectures for Parallel Processing*, McGraw-Hill Book Co., New York (in press to appear).

[8] Hwang, K., and S. P. Su, "VLSI Architectures for Feature Extraction and Pattern Classification," *Journal of Computer Graphics and Image Processing*, accepted to appear.

[9] Hwang, K., and Y. H. Cheng, "Partitioned Matrix Algorithms for VLSI Arithmetic Systems," *IEEE Transactions on Computer*, December 1982.

[10] Hwang, K. and Fu, K. S., "Integrated Computer Architectures for Pattern Analysis and Image Database Management," *IEEE Computer*, Jan. 1983 (to appear).

[11] Onoe, M., K. Preston, and A. Rosenfeld (Editors), *Real-Time/Parallel Computing: Image Analysis*, Plenum Press, N.Y. 1981.

[12] Preston, K., Jr. and L. Uhr (Editors), *Multicomputers and Image Processing*, Academic Press, N.Y. 1982.

[13] Preston, K., M. J. Duff, S. Levialdi, P. E. Norgren, and J. I. Toriwaki, "Basis of Cellular Logic with Some Applications in Medical Image Processing," *IEEE Proceedings*, Vol. 67, May 1979, pp. 826-856.

[14] Rosenfeld, A., "Parallel Processors for Image Processing: 2-D Arrays and Extensions," *IEEE Computer*, January 1983.

[15] Swartzlander, E. E., "VLSI Architecture," in *Very Large Scale Integration (VLSI): Fundamentals and Applications*, (Edited by D. F. Barbe), Springer-Verlag, New York, 1980.

[16] Kung, S. Y., et al., "Wave-Front Processors for VLSI Signal Processing," *IEEE Trans. Computers*, November, 1982.

[17] Yamamura, M., N. Kamibayashi and T. Ichikawa, "Organization of an Image Database Manipulation System," *Proc. Workshop Comp. Arch. for PAIDM*, Hot Springs, 1981, pp. 236-241.

[18] Yen, D. W. L., and A. V. Kulkarni, "The ESL Systolic Processor for Signal and Image Processing," *Proc. Workshop Comp. Arch. for PAIDM*, Hot Springs, Virginia, 1981, pp. 265-272.

Some Critical Issues in VLSI Signal Processor Implementation

Noble R. Powell*
Syracuse University
Syracuse, NY 13210

Abstract

This paper suggests that the critical issues pertaining to the very large scale integration of signal processor implementation are properly cast within the structure of the several design phases from concept to demonstration and test. It is further suggested, using a Matrix Functional Processor consisting of an array of LSI devices, that there are at least four critical issues supporting the contention that the signal processing community could benefit from revisions in educational programs, from improvements in design aids with extended data-bases, and from revisions in the traditional design processes. It concludes with an appeal for greater integration of the physical aspects extending and limiting signal processor performance into the conceptualization and comparative analysis of designs intended for implementation.

Introduction

The critical issues of VLSI signal processor implementation may be cast within the broad framework of the succession of design considerations that ultimately lead to the physical realization of a signal processing concept. The physical realization, or implementation, represents the result of a family of related balanced compromises made by the designers responsible for specific choices at each phase of the design process. The merit of the result and the design process depends sharply upon the extent to which those responsible for comprehensive superior completion of the implementation have developed an appreciation for the critical issues which arise at each stage of consideration in the steps required to pass from system concept to testing of the implementation. Depth and breadth

*Formerly at the Electronic Laboratory, General Electric Co., Syracuse, N.Y.

of understanding of the critical issues by the designers are the critical issues. This extremely demanding requirement of the signal-processing engineer will grow in importance as the variety of microelectronic alternative by which to implement signal processing concepts increases the means available to realize promising solutions to operational requirements.

Until the advent of very large scales of integration, the division of labor relating to the passage from system-concept to factory-test and repair could be effectively distributed in reasonably non-overlapping career categories or specializations. Since superior VLSI solutions to signal-processing problems so comprehensively span these traditional design-process slots in thinking and organization, a complacent compartmentalization or stratification(1) is no longer attractive. If effective and competitive application of VLSI to signal processing is to impact current practice in the context of standard, flexible, and custom componentry, the educational viewpoints, the design and instrumentation tools, and the design process will have to be modified. The modifications must reflect the need for increased emphasis upon the horizontal and vertical integration of design methodology related to signal-processor implementation. In such a spirit an outline of a few of the critical issues relating to improving the means available to realize VLSI solutions to signal processing problems follows. The perspective of the author is that these issues currently arise largely in the context of metal oxide semiconductors employed at very large scales of microelectronic integration for the purpose of signal processing -- a very limited scope, but one of some consequence.

The Framework of the Critical Issues

The natural basis for a discussion of the issues of importance pertaining to VLSI and signal processing implementation, the critical issue framework, is the set of major stages of effort leading to a physical realization. These may be regarded as the framework within which important unresolved problems of importance to the physical realization of comprehensive functional integration of signal processors may be cast. As such, they may be regarded as the basis for sketching the limits of consideration in structuring an educational viewpoint, a design tool, or a design process. There are ten overlapping but reasonably identifiable categories of importance with respect to which some critical issues of signal processor implementation may be cast. They are as follows:

Table 1.

Categories of Importance

1. System Definition
2. Architecture
3. Algorithmic Development
4. Numerical Analysis
5. Logic Network Design
6. Circuit Network Design
7. Topographical Description
8. Technological Utilization
9. Physical Innovation
10. Fundamental Extensions

These labels are intended to suggest categories of activity in terms of the principal consequences, or results, of a phase of effort in signal processor implementation. The process of design may legitimately commence and proceed from the point of inspiration in any of the ten categories. Thus, the critical issues of signal processor implementation may arise from the depths of any one category, or, from any set or combination of them, since all comprehensive applications require overlapping consideration of the ten categories. Further, it should be regarded as unlikely that any single category will overshadow any other insofar as the potential for significantly impacting the issues of implementation is concerned, in spite of the obvious merit of restricting consideration to subsets of the list for expediency.

If a criticism may be leveled at us in the signal processor community, it could be that these expedient restrictions have been so severe as to limit the set of admissible problems to those comfortably embedded within a highly restricted specialized category. It is the contention here that the significant advances in signal processor implementation in the future will not reside in the center-of-gravity of any of these categories, but rather in the consideration of some set or collection of ideas stemming from the creative marriage of the best ideas each category can offer at any given time. Thus, the discussion of the issues which follows is based upon the conviction that the solutions to contrived or expedient problems offer far less promise than the stuggle with ill-conditioned, incomplete, and relatively broad issues requiring the courage and effort to associate the controlling factors of one category with those of another. In a word, we are all well advised to struggle to broaden our respective data-bases in our search for superior solutions to the problems of signal processor implementation. The critical issues offered here are suggested in that spirit.

Some Critical Issues

The x-ware Issue

The initial concern arising in the context of attempting to match the power of VLSI signal processor solutions to operational requirements, product market-potential, or system compatibility is an issue with which a system-designer is confronted at the outset of his design. Faced with the need to specify a consistent set of functional transformations as a solution to meeting any operational requirement, the issue invariably involves the question of which transformations should be relegated to conventional off-the-shelf hardware, to software, to firmware, and to custom chipware; i.e., the x-ware issue. Clearly such an issue cannot be divorced categorically from any of the entries in Table 1, and cannot be considered settled except within the context of given time, resource, and operational constraints. The challenge for the system-designer is that of providing himself sufficient insight to each Table I entry so that he can make appropriate balanced judgements concerning the relative emphasis to be placed upon each x in his solution. Among the technical obstacles to this at present is the lack of adequate quantitative design aids and accessible data-bases which reflect the alternatives available for constructive comparative analysis. Since few problems lend themselves so readily to the use of custom VLSI as the problems of signal processing, it is imperative that consistent sets of adequate aids to design be developed.

A case in point is illustrated by the adjunct signal processor shown in Figure 1a and 1b depicting a Matrix Functional Processor block diagram and implementation, respectively. The signal processing system concept is cast squarely in the context of the x-ware issue since the operational requirement is to unburden host-processors having conventional organizations for arithmetically intensive linear-operator execution; namely, the solution of matrix functional equations arising in boundary-value problems, image filtering, adaptive filtering, non-linear optimization, and image construction from projections.

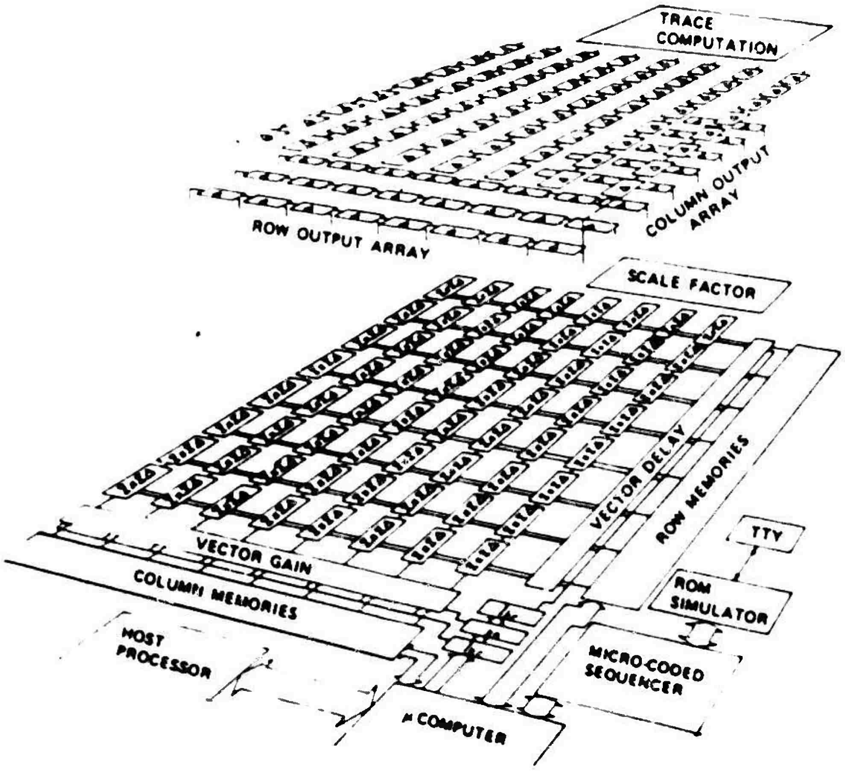


Figure 1a. Matrix Functional Processor

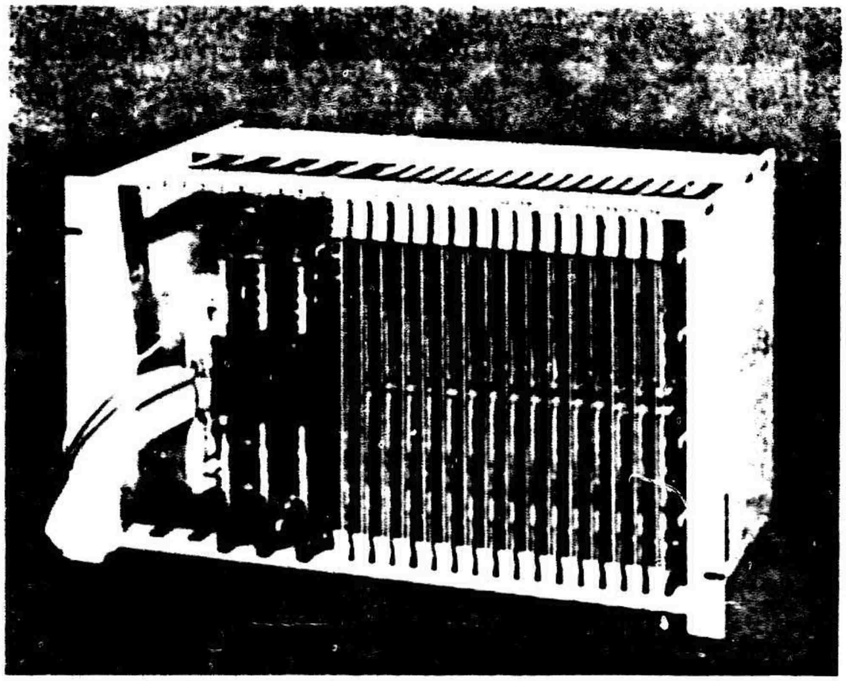


Figure 1b. 16 × 16 Matrix Functional Processor

It should be clear from the block diagram that the processor has two control components configured as a two-level hierarchy. On the higher level, a microprocessor controls the algorithm of matrix functions being performed, and on the lower level, a PROM driven sequencer controls the implementation of each matrix function, such as multiplication, transposition, inversion, etc. A host-processor supplies the data-matrices.

The VLSI x-ware issue arises here in the context of the design of the devices forming array-elements intended to provide the functional parallelism natural to the problem context and the VLSI technologies to be exploited. A balanced compromise is sought among the following desirable objectives:

1. At least two orders of magnitude of computational improvement over the current state of the matrix processing art at reduced cost.

2. A graceful means to consolidate the functional parallelism of the array directly at successively higher scales of solid-state integration as such consolidation becomes economically attractive.

3. A technology transparent means to future processor improvements in speed, power, and cost.

4. The flexibility to afford the user the means to execute such primatives as

$$S_N = S_{N-1} + R_{N-1} \cdot C_{N-1} ,$$

where R_{N-1}, C_{N-1}, S_{N-1} are conformable matrices of dimension kxm, kxn, nxm, respectively, with an array of dimension rxr, where r may be larger or smaller than k, m or n.

5. The elimination of recurring software costs at all levels.

6. A means of self-test and fault-tolerance.

These objectives suggest the rich variety of possible solutions to the x-ware problem and underline the importance of the need for simulation tools and accessible data-bases enabling early estimates to be made of the relative merit of x-ware assignments on a quantitative basis. The quantitative basis leads us to our next issue.

The Numerical Noise Issue

Any appraisal of the x-ware issue made out of context of the functional throughput and the processor numerical integrity must be considered a highly suspect partial solution. This is particularly the case in signal processing since the stochastic nature of the data processed defines the appropriate data transformations to be performed by the processor and, consequently, extremely careful attention must be given to the relationship between the transformations ideally required and the discrete or continuous versions of algorithms conceived to approximate the ideal process. It is seldom the case that the numerical methods appropriate for discrete realization have been analyzed in a context consistent with the means currently available to implement them in VLSI. For example, in the processor illustrated in Figure 1 the method of realization has been chosen to be a bit-serial concept.(2) This permits word-size to be dynamically variable under the control of the microprocessor as well as permitting block-floating point techniques to be used with double-precision scaling. The residual errors pertaining to matrix inversion, image generation, or various matrix decompositions are not generally available despite the great potential(3,4,5,6) for such numerical methods to reduce the effects of noise and the anomalies created by the nonlinearities of rounding, truncation, quantization, and limiting. These considerations suggest our next critical issue relating to implementation.

Technology Utilization Issue

The categorical span from 3 though 8 of Table 1 may be considered in the context of the question of utilizing a particular silicon technology in a manner which enables VLSI signal-processing to be performed in a cost-effective manner. This is not necessarily equivalent to utilizing a particular silicon technology in a manner which enables VLSI component manufacturers to benefit by the use of VLSI components in signal-processor designs. The former may imply considerable emphasis upon consolidation of function per chip, whereas the latter may imply considerable emphasis upon device density per chip and device-type proliferation. The issue is simply that of the proper balance between designing for signal processing system performance advantages, or for component utilization advantages. Each viewpoint has its merit in the context of the self-interest of the signal processor or the component supplier. The issue for us is how best to utilize the technology to produce superior signal processors. It is important to note that in either context it is essential to employ the technology closerto its ultimate capability, either to consolidate array functions as much as possible per device, or to render as competitive as possible the functionality per unit cost per device.

A case in point which emphasizes this distinction is the CE chip,(7,8) a nineteen thousand square mil device consisting of only 1330 transistors utilizing a 7.5 micron PMOS technology which implements a dot-product, $a \cdot b \pm c \cdot d$, to enable the concurrent execution of large complex-element matrix products with complex-element vectors. This device, motivated by the need to lower the incremental cost per signal-processing function in the frequency-domain, was completed at a time when a single multiplication operation could be completed only with a multitude of logically flexible, but functionally weak, off-the-shelf devices yielding signal-processing performance two orders of magnitude less attractive at significantly greater cost.

Rather large collections of computational elements have been suggested for use from an algorithmic viewpoint(9,10,11,12) to address the concerns of both the signal processing community as well as the component community regarding the potential value these arrays offer. It remains for all concerned to address the issue of appropriate technology utilization if the signal processing community is to obtain the specialized componentry they need instead of the componentry which, while addressing important computational problems, is mainly a consequence of the software intensive CPU markets. It now remains for this community to develop the concepts of technology utilization in the signal processing context, utilizing extended system design methods(12,13) where appropriate, to circumvent the limitations the technology may have to its effective and systematic use. This suggests the final critical issue to be discussed in the signal-processing/VLSI context here.

<u>The System Definition to Fundamental Extensions Gap</u>

Referring once more to Table 1, this issue relates to the significant separation that item 1 and items 9,10 seem to suffer on the one hand, in contrast to the extreme sensitivity the signal processing system implementation seems to have to physical innovations such as molecular beam epitaxy, laser-annealing, dry-etching, insulating substrates, III-V compounds, multilayer metal interconnect, and so forth. As a consequence of the potential such techniques offer for improving device performance, functional complexity per cell, and mixed analog and digital concepts of signal processor implementation, it is imperative that such advantages be reflected in concepts of hybrid analog-digital design and design aids. Monolithic sensor devices, microwave devices, and transducer devices can hardly be ignored in comprising a modern real-time signal processing system. An assessment of this potential will require careful revisions of what may now be regarded as fundamental limitations of current technology based upon linear models of basic ideas in solid state materials and electronic behavior. Unexploited effects stemming from a deeper understanding of micro, rather than macro, concepts of conduction and energy storage, via the routes of non-linear transport theory as well as the experimental means that monolayer epitaxy can provide may well be reflected in signal processing concepts highly materials-and-devices oriented. Layout considerations under such circumstances as well as network orientation of signal processing elements takes on prodigious proportions in such cases. Just as the circuit designer discovered that to be most effective he must work closely with the device designer in the '60s and '70s and now with the process engineer in the '80s, so may the signal processing engineer begin to avail himself more directly with new degrees of freedom -- freedom offered, for example, by tailoring the impurity profiles of his signal processing devices to his signal processing system requirements. The issue here is whether we develop the means appropriate to educate, to design, and to manufacture signal processors which will reflect these fundamental extensions.

Summary

We have suggested a framework for considering some critical implementation issues of very large scale integration of signal processors. Against such a framework we have suggested a few critical issues and suggested a rationale for their validity. Several concrete illustrations have been used to suggest the context of consideration or the basis for the issues. Briefly, they are the x-ware, the numerical noise, the technology utilization, and the systems-definition to fundamental-extensions gap issues. These issues suggest the need to reexamine our educational approach, our design-aid approach, and our design process procedures as we prepare for future signal processor implementation.

References

1. Mead, C., and Conway, L., Introduction to VLSI Systems, Addison-Wesley, 1980, p. 62.

2. Powell, N.R., and Irwin, J.M., "Signal Processing with Bit-Serial Word-Parallel Architectures," IEEE/SPIE Proceedings, Vol. 154, August, 1978.

3. Moyer, A.L., and Schlereth, F.H., The Design of Digital Leapfrog Ladder Filters, Eighth Asilomar Conference on Circuits, Systems, and Computers, December, 1974.

4. Thong, T., and Francis, J.E., A High Performance Hardware Graphics Generator, Society for Information Display Symposium, April, 1978.

5. Moyer, A.L., "A Parallel Matrix Inversion Algorithm for Dedicated Matrix Processor Applications," IEEE Joint Automatic Control Conference, October, 1978.

6. Thong, T., A Building Block for Digital Signal Processing: The DOA," Second IEEE International Conference on Circuits and Computers, September 29, 1982.

7. Powell, N.R., and Irwin, J.M., A MOS Monolithic Chip for High Speed Flexible FFT Microprocessors, IEEE International Solid-State Circuits Conference, February, 1975.

8. Powell, N.R., and Irwin, J.M., Flexible High Speed FFT with MOS Monolithic Chips, Eighth Asilomar Conference on Circuits, Systems, and Computers, December, 1974.

9. Kung, S.Y., VLSI Computing and Signal Processing Architecture, Conference on Advanced Research in VLSI, held at MIT, January 1980.

10. Kung, H.T., Systolic Arrays for VLSI in Sparse Matrix Proceedings 1978, Duff and Stewart, Chapter 8, Ref. 1.

11. Speiser, J.M. and Whitehouse, Parallel Processing Algorithms for Real-Time Signal Processing, IEEE/SPIE International Technical Symposium, August 25, 1981, Vol. 298.

12. Seitz, C.L.,"Self-timed VLSI Systems," Proceedings of the Caltech Conference on VLSI, January, 1979.

13. Lyon, R.F.,"A Bit-Serial VLSI Architectural Methodology for Signal Processing," VLSI-81, Edited by J.P. Grey, Academic Press, 1981

AD P002627

A Comparison of Arithmetic Units in VLSI Signal Processing Systems

by

Hassan M. Ahmed
Codex Corporation
20 Cabot Blvd.,
Mansfield, MA 02048

SUMMARY

Digital Signal Processing (DSP) is a term encompassing a variety of techniques for transforming digital samples of analog signals into samples of analog signals having more desirable characteristics. This paper is concerned only with DSP techniques which involve arithmetic in the usual sense, i.e., addition, multiplication, etc.

A VLSI digital signal processor may be viewed generically as in Figure 1. Most signal processing algorithms are very arithmetic intensive, often executing a set of structured operations repeatedly. Real time realization of many algorithms in present day VLSI signal processors is limited by the computation speed of these chips and the primary goal of research in the DSP area has been to enhance throughput via innovation in circuit speed, algorithms and architectures. Our view centers on the arithmetic unit in isolation, however we point out some simple architectural ideas in which the proper combination of arithmetic unit and controller provide remarkable throughput improvements.

In studying arithmetic unit design, our concerns are twofold. First, what are the appropriate arithmetic operations that are prevalent in signal processing algorithms? Second, what are efficient architectures for computing these operations?

Commercial digital signal processors are all predicated on the philosophy that fast multiplication and accumulation is basic to signal processing. (This is certainly true in many digital filtering algorithms.) Consequently, they all have a parallel multiplier and adder. We examine various ideas in multiplier design including bit serial and bit parallel approaches. Cellular multipliers are also examined due to their natural affinity for pipelining.

However, multiplication is just one in a set of fundamental operations which describe signal processing algorithms. Many adaptive filtering and matrix algebra operations require a fast vector rotation facility. The CORDIC technique provides a structured means for performing vector rotation and we examine various efficient implementations of the method. Convergence computation is the final technique we examine for computing useful elementary functions, and we show a common structure for realizing all three types of arithmetic units.

Not all signal processing algorithms are suited to a single type of arithmetic unit and we investigate some benchmarks which provide guidelines for the suitability of multipliers, CORDIC and convergence computation machines. For example, the u-law compander [1] used in many telephone applications is best suited to the convergence computation method and is difficult to implement with a multiplier alone. Alternately, a parallel multiplier provides the fastest realization of a tapped delay line filter [2]. Figure 2 shows some benchmark performance figures for typical machines exhibiting the three types of arithmetic units considered.

REFERENCES

[1] J.R. Boddie et al, "Adaptive Differential Pulse Code Modulation Coding, "Bell Systems Technical Journal, Vol. 60, No. 7, September 1981

[2] Oppenheim, Schafer, DIGITAL SIGNAL PROCESSING, Prentice-Hall, 1975

[3] Ahmed, "Signal Processing Algorithms and Architectures", Ph.D Dissertation, Dept. of Electrical Engineering, Stanford University, June 1982.

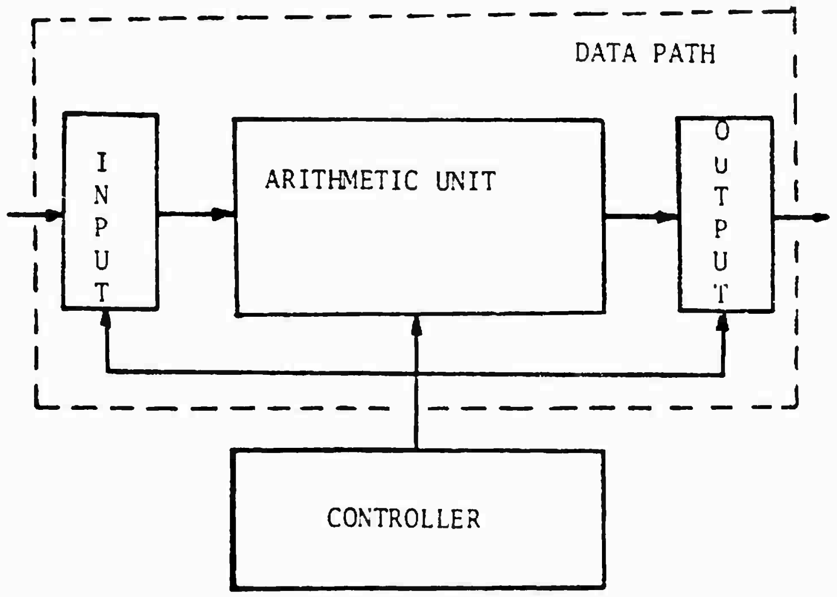


Figure 1. Generic Digital Signal Processor

	EXECUTION TIME		
ALGORITHM	PARALLEL MULTIPLIER	CORDIC	CONVERGENCE COMPUTATION
128 tap FIR FILTER	.13 ms	.25 ms	-
SINE/COSINE	5.3 us	2.2 us	2.5 us*
uLAW COMPANDER	1 us with table lookup	4 us	1 us
32 point COMPLEX FFT	0.7 ms	.5 ms	0.5 ms*

Sample rate = 8 KHz

* - indicates complex version of convergence computation method [3].

Figure 2: Benchmark Programs

A SYSTOLIC ARCHITECTURE FOR THE LINEAR LEAST-SQUARES AND EIGENVALUE PROBLEMS

by

Robert Schreiber
Philip Kuekes

Certain modern high-resolution data-adaptive techniques for spatial direction and power spectrum estimation require the very high-speed solution of matrix least-squares problems and eigenvalue-eigenvector problems. The standard numerical methods for these problems are QR-factorization for least-squares and bandwidth reduction followed by QR-iteration for eigensystems.

The highly structured data flow of these algorithms makes them ideal for computation by systolic arrays; with n-column matrices, $O(n^2)$ parallel processors can be effectively used.

We first discuss QR factorization of full, rectangular, complex matrices. After introducing an efficient method based on plane rotations, we give a triangular systolic array design using a "cell" for complex plane rotations. This cell can be implemented using six identical VLSI chips. We also discuss implementation of a suitable boundary cell that generates rotations. Alternative methods, based on factoring real matrices of twice the size, are contrasted with the one chosen.

In practice, a trapezoidal subarray would be used. We discuss the problems of decomposing the problem for solution by a subarray, unloading the elements of R from the array, and connecting the array to temporary storage.

For least-squares problems, the same array may be used for the most time-consuming tasks; reduction of a full Hermitian or general complex matrix to banded or upper Hessenberg form. The subsequent QR iteration can be done by rectangular arrays using nearly identical cells.

AD P002628

THE MICROELECTRONICS CENTER - A NEW FORCE IN COOPERATIVE VLSI SIGNAL PROCESSING RESEARCH

Richard B. Fair
Microelectronics Center of North Carolina
Research Triangle Park, North Carolina 27709

I. Introduction

Over the past few years the use of silicon as an implementation medium by researchers in signal processing has increased dramatically. This is due mainly to a simplification and rationalization of the interface between IC designer and the fabrication medium. As a result, new architectures and design methods have flourished to support a new breed of IC designer. Focus to date, however, has been at the chip level and below. Certainly, attention has been focused on the architecture of large systems that do signal processing with VLSI as a medium, but few, if any, have been built.

The real world constraints brought about by a desire to implement sophisticated signal processing systems in silicon impacts the design tools, fabrication tools, and associated support. Consequently, there exists a real need to pull together resources that can collaborate in such a way as to provide the needed ingredients for vertically integrated programs. It is very difficult for universities alone to provide this support considering the extensive resources needed to mount a program committed to VLSI fabrication and design. In universities we often find innovative design tools and software of a "proof-of-principal" nature. The intent is to pass ideas to government and industry so that their endeavors might be aided. Now this does not happen rapidly because of inertia that exist in large commercial operations. In addition, most universities are never given the opportunity to close the design loop - that is, to find out how good their design tools really are. Likewise, the system designer is rarely given the opportunity to put his designs into silicon except to interact with a foundry operation. Unfortunately, the concept of the foundry is a snare and a delusion. It can't provide a good product when it is loaded down with many custom parts requiring rapid turn around and low volume. This type of operation can frustrate system designers who may need large numbers of tightly specified chips to build signal processing machines.

The needs of the microelectronics industry in applying VLSI to modern signal processing are:

1. To innovate new designs at a rapid rate,

2. To maintain an edge on the numbers of new designs and innovations,

3. To reduce the amount of parallel research and development efforts required to make effective technology and design decisions and to achieve successful market dominance.

In order to achieve these goals in the face of rising costs of research and design time, there are four principal means of promoting research and development and of extracting useful results at an early stage:

1. Joint R&D ventures among firms;

2. Competition between firms working in parallel;

3. University research or autonomous centers doing non-profit basic research; and

4. Semi-autonomous centers of non-profit research such as microelectronics research centers tied in with universities with some funding and direction supplied by private firms.

The needs of the federal government are to design and develop advanced integrated circuits that will be based upon military needs, such as signal processing. In order to muster significant resources the VHSIC Program was initiated by DOD to solicit the support of universities and industry. Thus, this program can be viewed as a cooperative effort among industry, university, and government to impact primarily in the area of military capability. However, this effort cannot be construed as guiding the entire microelectronics industry's technical and commercial future. And, the imposition of information control has been viewed by the university community as an impediment to doing anything beyond basic science.

In actuality therefore, the needs of industry, government and the universities are really quite different and many times are in conflict with each other. For example, the competition for highly skilled professionals to work in VLSI design and technology development is very intense. University faculty and students who are capable of doing advanced VLSI design are snatched up by industry with salary offers that the universities cannot match. By depleting the intellectual capital in the universities and by having only meager facilities available, there may be little incentive for industry to take seriously the offer of collaborative research opportunities offered by the universities.

II. Opportunities for Cooperative Research Ventures

In the previous section we saw several potential ways in which cooperative research ventures can impact on VLSI and modern signal processing research. From industry's point of view it has recently been suggested that a joint research venture be initiated called the microelectronics and computer technology enterprise. The cost associated with negotiating such a joint venture could be prohibitive. Agreements would have to be achieved over research objectives, manpower issues (who could do what), patent rights, information sharing, and at what point can a firm begin to maneuver to obtain the first mover advantages. In addition, this organization would not impact on the nation's educational needs in VLSI.

Sponsored university research alone provides one solution to the joint venture contracting problem. At least firms then have a measure of protection against opportunistic behavior by university scientists. However, the basic research conducted generally produces information not intended for short term commercial gains. Thus, between the highly focused R&D of industry and unstructured university research lies the gray area of research projects involving technological issues of potential, but unproven commercial significance. The projects include new computer-aided design tools, new signal processing architectures, and new processing technologies which may or may not reach commercial status in the long run. However, this type of research is in the public interest since it significantly aides industry in deciding where innovation is or is not likely to be profitable and, therefore, where precious capital resources are best invested.

III. The Microelectronics Research Center - A New Partner In Joint Research Ventures

A new alternative in joint research ventures is the concept of a not-for-profit microelectronics research center (MRC). The MRC can be integrated into the universities' research areas and still deal with the industrial component of experimental R&D. The role of the MRC would be to vertically integrate basic semiconductor research produced in a university community with advanced processing development. The processing work would provide a basis for continually updating a baseline wafer fabrication facility for producing experimental integrated systems chips. A parallel path for computer science exists leading to advanced design methodologies and new system architectures. Thus, industrial investment and cooperation through MRC's offers not only potential economics to firms from the shared purchase of equipment and training of prospective employees, but also provides an efficient organization for funneling basic research results produced in the universities into potentially useful technologies. It is envisioned that firms sponsoring cooperative research funneled through MRC's would be given early access to research results. In addition, employees from member firms could work in the MRC's with faculty and graduate students, and thus affect efficient technology transfer. However, the bulk of the work would not be considered proprietary to any given firm. Thus, a sponsoring firm's ability to obtain a competitive advantage in a particular area would depend on its own inertia to innovate towards a final product. The role of the MRC would be to conduct integrated systems research, produce research results in electronic materials and devices, and would evaluate technological choices. Evaluation of the eventual commercial response to such choices still remains as the risk that will be incurred by industry. The role of the federal government in this joint venture would be largely catalytic. By supplying funds to research projects that lie within the scope of the MRC, the government can provide the initiative to university participants that is required.

The concept of the MRC is, then, an organization that will benefit the microelectronics industry, government, as well as the universities with their educational and research programs. For industry the MRC is a response to the need for increased manpower, increased speed of innovation, and more effective technology decision making processes. For the federal government, the MRC provides an organization that can produce a vertically integrated design capability that involves itself with the basic research community of the universities. For the universities the MRC can serve to enhance significantly their educational and research programs. In turn, the MRC must depend upon the university community for a portion of its resources such as people, equipment, and collaborative programs. In the same measure, faculty members who seek to do advanced microelectronics research will be incomplete in the resources available on campus, considering the very large cost of equipment and clean lab space. Thus, a mutual dependence for the common good must exist.

IV. The Microelectronics Center of North Carolina - The First MRC

The research goals of the Microelectronics Center of North Carolina (MCNC) include unstructured campus research in electronic materials, devices, and computer science as well as focused research in silicon processing, integrated devices, and the architecture and design methodology of silicon systems. The goal of silicon research will be to vertically integrate materials work with advanced processing development. The processing results will provide a basis for continually updating a baseline wafer fabrication facility which will produce experimental integrated systems chips. This vertical integration of research and development programs will speed the implementation of laboratory results into useful processes or designs. Additionally, the coupling of process-related research and systems design research will drive each respective area towards relevance and recognition of the particular needs of industry.

A major thrust at MCNC has been the development of industrial quality design tools. At the present time the design tools at MCNC involve the use of symbolic "assembly level" tools which ease the circuit designer's access to correct masks. This approach is based on the abcd language. Work is being carried out in parallel on systems level "targeted silicon compilers" for high-level optimization of certain architectures. A key to these methods is the use of floor-plan-driven architecture and functional simulation. Software is written and supported by professionals.

MCNC also plans to have its own fabrication and mask making facilities that are designed to support architectural systems research. The "baseline process" will support a state-of-the-art process aimed at the system designer and supported by a full-time staff of professionals. We expect that at any point in time we would have a process between that available in research labs and that available "over the counter" at custom houses. A goal of the process will be to provide a friendly interface for ease of circuit design, tool design, and high yield. Note that the goals of this process line are not consistent with those normally found in the "piece part" industry, but rather more in line with systems companies; that is (small runs, high cost per dye). It is expected that electronic materials and silicon processing research carried out at the participating institutions will support this baseline process. This vertical integration of research and development programs is substantially different to that found in universities alone. The goal of this program is seen as being able to train students to be aware of the benefits of silicon as an architectural medium in signal processing. As the second goal we would hope that the work at this level would yield tools and methods to aide designers in selecting architectures and in evaluating them for a given problem. All the time we strive to seek a balance between high-level design while retaining some notion of the power of silicon. We would also hope that feedback to the material scientists would ultimately yield new processes that directly benefit the system designer.

Thirdly, by a combination of the two goals mentioned, we would like to have a friendly silicon facility where a potential user can enter at the level at which he or she feels most comfortable and exit with his or her desired product - be that a new device or a new real time flight simulator. The impact of this approach on using VLSI in signal processing architecture is obvious.

MCNC is being designed to provide rapid implementation of experimental architectures into silicon systems through a vertically integrated support base of design tools, silicon processing, and system assembly. The impact of this MRC will be significant to the educational and research goals of the participating institutions as well as to the needs of its supporting institutions - industry and the Federal and North Carolina State Governments. From the State's view point, the favorable ingredients that have gone into MCNC will also form the foci for considerable industrial growth. From the Federal Government's view point, they only need to interface with one umbrella organization for funding that draws synergistically on the integrated resources of six other institutions. From industry's view point, they can interface with university research through a new organization that can speed the development and evaluation of new technologies, and that can help decide where precious capital resources are best invested.

Computing Theory and Practice

Richard L. Lau
Office of Naval Research

This note is somewhat tangential to the theme of the corresponding panel session, but it is presented in the belief that the evolution of the relationship between theory and practice in computational sciences is crucial to the healthy development of these sciences and the corresponding engineering practice. It seems appropriate to consider these matters at the beginning of a new discipline.

Many subdisciplines in computer science research or practice have formed the basis for theoretical work which is aimed at shedding light on the corresponding "applied" work. This theoretical work is distinguished by the use of mathematical tools or modes of thinking. This essay will discuss both the benefits and pitfalls of theory, with an emphasis on pitfalls. It is written from the perspective of one who is charged by the Navy to support research which is both excellent science and which is likely to have an impact on important Navy problems. The search for combinations of these qualities leads one to be rather impatient with certain endeavors. There will be a tendency here to apply somewhat pejorative terms to theoretical work with little impact on practice. The point of view will be that of one who is seeking a payoff from research in the medium term.

Generalities

Good generalizations promote the condensation of knowledge; bad generalizations promote fragmentation. Good theory allows us to know more and remember less. It gives those of us in the trees a view of the forest.

The degree of generality (or abstractness) tends to increase in the absence of pressures toward relevance to the real world. When a given area of theory becomes played out, one can often generalize and open up an entirely new area for research, one that is likely to have even less relevance to practice than the old one.

Excessive levels of abstraction promote the worship of cleverness for its own sake and the devaluation of practice.

I will illustrate the last point with a true incident of recent vintage. Names have been changed to protect the guilty. About a year ago Prof. Good stopped by my office to report on a new algorithm he had developed for problem P. P is a problem which was pushing the existing algorithm/machine combinations to their limits; much larger versions of P loomed on the horizon. Failure to find a better methodology for P would have had a serious practical effect. Good developed an interesting new way to characterize problem P. Given that characterization, it was moderately easy to develop and analyze the new algorithm (i.e., blinding cleverness was not displayed). The new algorithm is a couple of orders of magnitude faster than previous algorithms.

Some months later Prof. Innocent, chairman of the very good computer science department at Ivy University, phoned me to say that his department would like to hire someone in algorithmic studies. I suggested that they invite Good for a seminar. That was done and Good talked about his work on problem P. Innocent phoned to say that the Ivy theorists were taken aback by Good's talk and were heard to be mumbling "hacking".

Shortly thereafter, I was speaking with Prof. Adept, a very important figure in complexity, whose research has been influential on both theory and practice, not because he is very interested in practice, but because his results have been of such width and depth that they have, ipso facto, influenced practice. During our conversation Good's name come up, and Prof. Adept said that much of Good's work is good, but he has a tendency to hack on practical problems. He cited Good's work on Problem P. Still later, I was speaking to Prof. Wise whose taste and judgment in algorithmic matters (both theoretical and applied) is universally acknowledged to be impeccable. We fell to discussing who he considered to be the top four or five practitioners of computational complexity. Good was on that list! QED.

A look at the history of numerical analysis and of computational complexity over the past fifteen years suggests a sociological paradigm for the evolution of the balance of theory and practice in relatively new fields. Numerical analysis and computational complexity are "theoretical" subjects, with their results aimed at computational physics and algorithm implementation respectively.

I first started looking at these two subjects in the mid-to-late sixties, at which time complexity research was just getting off the ground but numerical analysis had been on active field for many years. In the late sixties numerical analysis underwent a major change. Many of the best people in the field transitioned from theorem proving to algorithm development. That movement has continued to the point where many of the most respected numerical analysts consider convergence proofs of interest only if they directly support work on algorithms.

I believe that numerical analysis is right now going through another transition from the general to the particular. Whereas algorithms set the tone for the seventies, the flavor of the eighties will, to some extent at least, be a concentration on "real world" problems in computational physics. Numerical analysts will no longer treat mainly simple, general problems with many of the difficulties assumed away. Part of the reason for this transition is that developments in numerical analysis have provided the tools for attacking hard problems. This development, coupled with the community pace setters' perception that numerical analysis is interesting to the degree that it is useful, is driving the transition activity.

From my limited perspective, in the past decade or so computational complexity has grown to be increasingly abstract, with many of its models having little to do with "real" computing, and much of its work applicable only in asymptotia. Prof. Good tells of speeding up an important algorithm in Prof. Smart's book by a factor of several hundred. Smart could not see the point if the big-O exponent didn't change. This is not to say, however, that theoretical work is not valuable - the best of it gives perspective.

I believe that I see the first signs of a change in the complexity world analogous to the late '60's change in numerical analysis. A number of the most capable people (i.e., the old boy network) are turning to algorithm work of immediate interest for applications. The sorts of people making this transition are important; they are a fairly large segment of the "taste makers". This exactly parallels the numerical analysis case. The same mechanism may have been at work in both subjects. When a subject is in its infancy theory can be very important because it can place the subject in a broad context, it can shed light on which areas are likely to be most fruitful, and it can be a guide to the practitioner (if he understands its practical application - which is often not the case). Subsequently, theory may take on a life of its own, and, as different "tastes" develop, continuously draw away from practice.

If the numerical analysis experience is predictive, we may look forward to further movement in the direction of practice. In the hope that the leaders of the emerging field of architecture appropriate for VLSI implementation may wish to avoid too great a gap between theory and practice, I have some suggestions for avoiding a schism or minimizing its impact.

Remedies

Theoreticians should be encouraged to mix with practitioners. They should not form their own theoretical societies.

The old boy network should declare an end to some segments of theory when they become theory for the sake of theory, i.e., have no further light to shed on practice.

Referees should be ruthless and relentless (Parlett's Rule).

Parlett's Rule is so designated in recognition of the sometimes heroic means he employs in his mission to prevent bad papers from being published - at least in the better journals. Some time ago I received a very long paper from Prof. O'Blivious, a well know and widely respected computer scientist. I could understand both the definitions and the theorems, but I could not see the point of the paper, i.e., how the result fitted into existing knowledge in the field. I sent the paper to Parlett to see if he could shed light. His reply said that he was writing at 1:00 am, he had been looking at the paper since 10:00 pm, and he finally had seen through the somewhat baroque definitions to the heart of the matter. The main result was a minor perturbation of existing knowledge. Its likely role, if published, would be to thoroughly confuse graduate students.

Parlett communicated his criticism to O'Blivious, with no result. He then did the ultimate in refereeing, an act which prompted me to formulate Parlett's Rule. He strengthened the main result of O'Blivious's paper, and proved the stronger result (which had taken O'Blivious fifty pages) in two pages. Parlett then communicated the result to O'Blivious with a note saying that the strengthened result was mildly interesting, but that he didn't think it worthy of publication.

←

AD P002680

VLSI SIGNAL PROCESSING - BASIS FOR COOPERATION

L.W. Sumney, Executive Director
Semiconductor Research Corporation

In March 1981 I participated in a DoD Special Technical Area Review on Signal Processing. Beam forming in phased array antennas and sonar, digital spectral analysis for electronic warfare, and target classification were cited as the major needs pushing the device technology to ever higher throughput rates. The need for these higher throughput rates is one of the major drivers for the VHSIC Program. The question that ~~we might~~ pose is whether there are other applications of signal processing, i.e., non-defense applications that can provide a basis for cooperation among the identified VLSI communities of interest. The answer to this question requires a brief examination of the VLSI-signal processing relationship.

VLSI is providing a capability for the design and implementation of systems on silicon chips. Application of this capability to conventional algorithms and architectures will be significant but much greater gains can be realized when research into algorithms, architectures, and VLSI technology are applied together to the realization of optimum signal processing systems. Already, highly concurrent algorithms for signal processing are being developed for digital filters, linear equations, eigenvalue extraction, and other equally demanding tasks. These highly concurrent structures are being designed to take advantage of VLSI, their computation speed is linearly dependent on the problem order and many elemental processors are required.

The large potential improvements in processing speed offered by concurrent architectures require a more complex, although highly regular, system architecture. As a consequence, built-in-test (BIT) and fault tolerant designs are required in order to assure acceptable system reliability. For example, the failure of a processor at a point deep within an array of processors must be detected quickly and corrective action taken if VLSI with concurrent architectures are to obtain broad application.

Beyond the realm of classical signal processing functions lie higher level functions such as those associated with machine intelligence.

Intelligent machines will require extraction of much more information from the signals received than provided by classical methods. Recognition, extrapolation, association, and prediction algorithms are examples. Because of the newness of the field, little work has been reported on efficient architectures for intelligent machines, but the potential has been recognized. For example, the fifth generation computer project in Japan is emphasizing machine intelligence.

This brief review of some aspects of the signal processing and VLSI synergism allows us to respond to the first question. There are a number of non-defense applications of high speed signal processing that can serve as the basis of industry and university cooperation in signal processing. While intelligent machines may be the most important of these in the long haul, there are other applications with greater near-term importance. Real-time feature extraction for on-board remote sensing systems is of great importance for bandwidth reduction in satellite communications systems as well as for medical diagnosis in real time high-resolution tomography. These two applications are representative of systems that will derive information from three dimensional objects or scenes to establish a higher order of information than is now possible. In a similar mode, other systems will require complex processing of written or printed text, speech, and other data streams in order to translate, recognize, or encrypt. The correlations required in such operations will require dealing with spectral and temporal relationships over large data bases in real time. All of us who have been faced with the language translation problem in our academic careers can appreciate the nature of the signal processing tasks required to change spoken or written German into intelligible English. The complex relationships in strings of text and the large data bases stretch the capabilities of many students.

Thus, although signal processing has been looked at primarily as an imperative of the defense department, there are a wide variety of future non-defense needs that will require equal signal processing capabilities, and thus the basis for cooperation exists. One form of cooperation that is now being implemented lies in the area of generic semiconductor technology and is the province of the Semiconductor Research Corporation (SRC).

SRC is itself a cooperative effort of the semiconductor industry, integrated circuit users, and equipment/materials suppliers to the industry to address needs in generic technology and in education that are important to the future stance of the U.S. semiconductor device industry. SRC members provide funds which are being applied to the implementation of an integrated research effort in American universities. SRC is still young, our initial research activities are just now getting underway, but within a year the level of research support provided through SRC will be approaching $10 million. As its program becomes established SRC expects to cooperate with the various government and private efforts with similar interests. Initial contacts have been made with NSF and DoD to effect coordination and cooperation. As we move ahead it is necessary that the investment in and productivity of the U.S. research effort in VLSI and signal processing be maximized so as to enhance the leadership position that we now enjoy.

AUTHOR INDEX